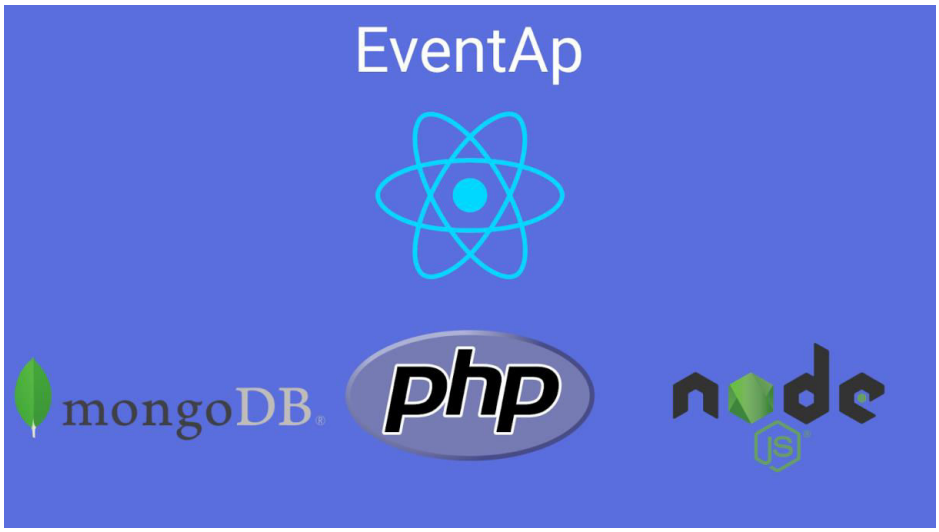
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη cross platform (android & iOS) εφαρμογής για την προβολή και τη διαχείριση εκδηλώσεων



Του φοιτητή
Παναγιώτη Γεώργιου Τσιμπουκέλλη
Αρ. Μητρώου: 164766

Επιβλέπων
Ευκλείδης Κεραμόπουλος
Αναπληρωτής Καθηγητής

Ημερομηνία 14/06/2021

Τίτλος Δ.Ε. Ανάπτυξη cross platform (android & iOS) εφαρμογής για την προβολή και τη διαχείριση εκδηλώσεων

Κωδικός Δ.Ε. 20113

Ονοματεπώνυμο φοιτητή Παναγιώτης Γεώργιος Τσιμπουκέλλης

Ονοματεπώνυμο εισηγητή …

Ημερομηνία ανάληψης Δ.Ε. 06/04/2020

Ημερομηνία περάτωσης Δ.Ε. …

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Ο λόγος που επέλεξα τη συγκεκριμένη πτυχιακή ήταν για να έρθω σε επαφή με δύο διαφορετικά λειτουργικά συστήματα κινητών συσκευών (android & ios) και μέσα από την ανάπτυξη της εφαρμογής να δω τις απαιτήσεις, τα πλεονεκτήματα αλλά και τα μειονεκτήματα του κάθε λειτουργικού, μέσα από τη σκοπιά της React Native. Επιπλέον λόγω της δημιουργίας και διαχειριστικού περιβάλλοντος είχα την ευκαιρία να έρθω σε επαφή με την PHP, αλλά και με node.js καθώς για τη διασύνδεση της βάσης δεδομένων με τις εφαρμογές δημιούργησα ένα API Service. Τέλος επέλεξα την mongoDB για Βάση Δεδομένων για να μάθω καλύτερα πως δουλεύει μια no sql βάση.

Περίληψη

Αντικείμενο της παρούσας πτυχιακής εργασίας αποτελεί ο σχεδιασμός, η ανάπτυξη και η υλοποίηση μιας cross platform mobile εφαρμογής προβολής εκδηλώσεων με χρήση της React Native, αλλά και μιας διαδικτυακής πλατφόρμας διαχείρισης εκδηλώσεων με τη χρήση της PHP. Η βάση δεδομένων υλοποιήθηκε σε MongoDB και η διασύνδεση των εφαρμογών με τη βάση έγινε εφικτή με την ανάπτυξη ενός API, έτσι ώστε να υπάρχει μεγαλύτερη ασφάλεια.

Development of cross platform (android & iOS) application for event booking and management

Panagiotis Georgios Tsimpoukellis

Abstract

The aim of this diploma thesis is the design, development and implementation of a cross platform mobile event application using React Native and also an online event management platform using PHP. The database was implemented in MongoDB and the interconnection of applications with the database was made possible by developing an API, so that there is more security.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

UI	User Interface
NPM	Node Package Manager
API	Application Programming Interface
CSS	Cascading Style Sheets

Κεφάλαιο 1ο:Εισαγωγή

1.1 Εισαγωγή στη React Native

Τα τελευταία χρόνια το μερίδιο της αγοράς που κατέχει η ανάπτυξη εφαρμογών για κινητές συσκευές έχει αρχίσει να αυξάνεται με ραγδαίους ρυθμούς. Υπάρχουν εταιρείες κολοσσοί που έχουν αναπτύξει τα δικά τους frameworks για τη δημιουργία mobile εφαρμογών. Μία από αυτές είναι η Facebook, η οποία λάνσαρε τη React Native.

Η React Native είναι ένα framework ανοιχτού κώδικα για την ανάπτυξη εφαρμογών για κινητές συσκευές. Παρουσιάστηκε στο κοινό τον Μάρτιο του 2015 από τη Facebook. Το συγκεκριμένο framework επιτρέπει στους χρήστες να αναπτύξουν cross platform εφαρμογές σε android και iOS.

Η ανάπτυξη αυτών των native εφαρμογών γίνεται με τη χρήση της React, που και αυτή αποτελεί δημιουργία της Facebook (κυκλοφόρησε το 2013) και χρησιμοποιεί από πίσω Javascript. Το βασικό πλεονέκτημα της React είναι η γρήγορη και εύκολη δημιουργία UI (διεπαφής χρήστη) με τη δημιουργία επαναχρησιμοποιούμενων components. Το κάθε component έχει το δικό του state και τα δικά του properties. Δηλαδή κατασκευάζουμε ένα γενικό interface και στη συνέχεια τα components διαφοροποιούνται αναλόγως με το state και τα properties.

Η παρούσα πτυχιακή εργασία παρουσιάζει την ανάπτυξη μιας εφαρμογής για εκδηλώσεις.

1.2 Στάδια υλοποίησης εφαρμογής με React Native

Η συγκεκριμένη εφαρμογή έχει υλοποιηθεί σε React Native[6] με τη χρήση της βιβλιοθήκης Expo.io ώστε να υπάρχει καλύτερος έλεγχος και μεγαλύτερη ταχύτητα ανάπτυξης, καθώς πολλά εργαλεία που συμπεριλαμβάνονται στην expo βοηθούν σε αυτό το σκοπό.

Επιπλέον ανέπτυξα μια web εφαρμογή σε PHP[4] (Backend) και Bootstrap (FrontEnd) με σκοπό να μπορούν οι διαχειριστές των εκδηλώσεων να επεξεργαστούν και να προβάλλουν τις εκδηλώσεις τους.

Πέρα από την εφαρμογή για κινητές συσκευές, θα αναλύσουμε την δημιουργία μίας διεπαφής προγραμματισμού εφαρμογών (Application Programming Interface) API[9]. Το συγκεκριμένο Web API δημιουργήθηκε με node.js και σκοπό έχει την ασφαλή ανταλλαγή δεδομένων μεταξύ της mobile εφαρμογής και της βάσης δεδομένων αλλά και της web εφαρμογής για τη διαχείριση των εκδηλώσεων και της βάσης δεδομένων.

Η βάση δεδομένων επέλεξα να είναι κάποια η οποία θα είναι μη σχεσιακή (no sql) γι αυτό κατέληξα στη mongoDB[10] καθώς από την έρευνα που έκανα διαπίστωσα ότι είναι η ταχύτερα αναπτυσσόμενη τεχνολογία για βάσεις δεδομένων και μετρά ήδη πολλές εγκαταστάσεις σε εφαρμογές που χρησιμοποιούν δισεκατομμύρια χρήστες.

Τέλος, ο server στον οποίο ανέπτυξα το web API και ο server στον οποίο ανέπτυξα τη web εφαρμογή για τη διαχείριση των εκδηλώσεων σε PHP, φιλοξενούνται στη Hetzner, στην οποία καταβάλλω ένα πολύ μικρό αντίτιμο.

1.3 Δομή της εργασίας

Η δομή της εργασίας συνοψίζεται στις εξής ενότητες:

- Στο πρώτο κεφάλαιο γίνεται μια γενική εισαγωγή στην ανάπτυξη mobile εφαρμογών με React Native.
- Στο δεύτερο κεφάλαιο αναλύεται ο τρόπος σχεδιασμού, υλοποίησης και ανάπτυξης της εφαρμογής.
- Στο τρίτο κεφάλαιο γίνεται εκτενής ανάλυση του User Interface τόσο της mobile όσο και της web εφαρμογής
- Στο τέταρτο κεφάλαιο αναλύεται τόσο θεωρητικά όσο και πρακτικά η δημιουργία ενός ubuntu server
- Στο πέμπτο κεφάλαιο παρουσιάζεται η βάση δεδομένων, το σχήμα της και ο τρόπος δημιουργίας μιας βάσης δεδομένων σε MongoDB
- Στο έκτο κεφάλαιο γίνεται εκτενής ανάλυση στην πρακτική δημιουργία του REST API
- Στο έβδομο κεφάλαιο παρουσιάζεται αναλυτικά η δημιουργία της web εφαρμογής διαχείρισης εκδηλώσεων
- Στο όγδοο κεφάλαιο παρουσιάζεται αναλυτικά η δημιουργία της mobile εφαρμογής προβολής εκδηλώσεων
- Τέλος, στο ένατο κεφάλαιο γίνεται μια σύνοψη όλων των παραπάνω, παρουσιάζονται δυσκολίες που αντιμετωπίστηκαν αλλά και επεκτάσεις που μπορούν να προστεθούν στο μέλλον

Κεφάλαιο 2ο: Διεπαφή Χρήστη

2.1 Εισαγωγή

Βασικός παράγοντας στην ανάπτυξη μιας εφαρμογής είναι ο σχεδιασμός και η εμπειρία που θα αποκομίσει ο χρήστης. Για τον σκοπό αυτό έκανα μια έρευνα σε ήδη δημοσιευμένες εφαρμογές και τον τρόπο που είναι δομημένη η διεπαφή χρήστη.

Ένας από τους κύριους άξονες είναι η εμπειρία του χρήστη και πόσο εύκολα μπορεί να καταλάβει μόνος του πως να χρησιμοποιεί μια εφαρμογή χωρίς τον εκπαιδεύσει κάποιος ή να διαβάσει κάποιο documentation.

Η έρευνα σε ήδη υπάρχουσες εφαρμογές είναι το πιο σημαντικό κομμάτι, καθώς σε αρκετές περιπτώσεις μπορούμε να κατανοήσουμε καλύτερα τον τρόπο με τον οποίο οι χρήστες αντιλαμβάνονται και αλληλεπιδρούν με κάτι. Βέβαια όλα αυτά έχουν να κάνουν με το προφίλ του χρήστη στο οποίο απευθυνόμαστε. Για παράδειγμα διαφορετικά θα είναι σχεδιασμένη και υλοποιημένη μια εφαρμογή η οποία απευθύνεται σε έναν έμπειρο χρήστη της τεχνολογίας και διαφορετικά μια η οποία απευθύνεται σε κάποιον ο οποίος έρχεται ίσως για πρώτη φορά σε επαφή με την τεχνολογία.

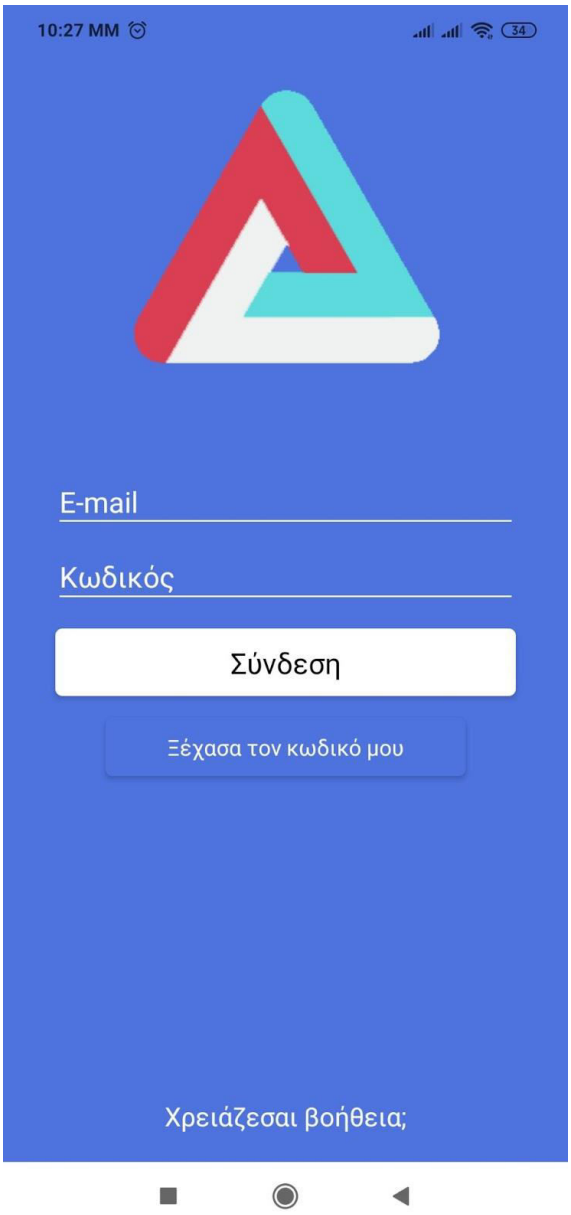
Για τη συγκεκριμένη πτυχιακή εργασία κρίνουμε ότι η εφαρμογή θα χρησιμοποιείται από χρήστες με διαφορετικό εύρος γνώσεων άρα θα πρέπει να τείνουμε στο να είναι εύχρηστη για τους απλούς χρήστες που θέλουν εύκολα και γρήγορα να κλείσουν μια εκδήλωση.

Αντίστοιχα εύκολο θα πρέπει να είναι και για τους διοργανωτές στο web περιβάλλον να δημιουργήσουν μια εκδήλωση. Με την έρευνα που έκανα σε web templates, κατέληξα σε Bootstrap Template, ενώ για την εφαρμογή για κινητά δημιούργησα δικές μου οθόνες.

2.2 Wireframes Mobile App

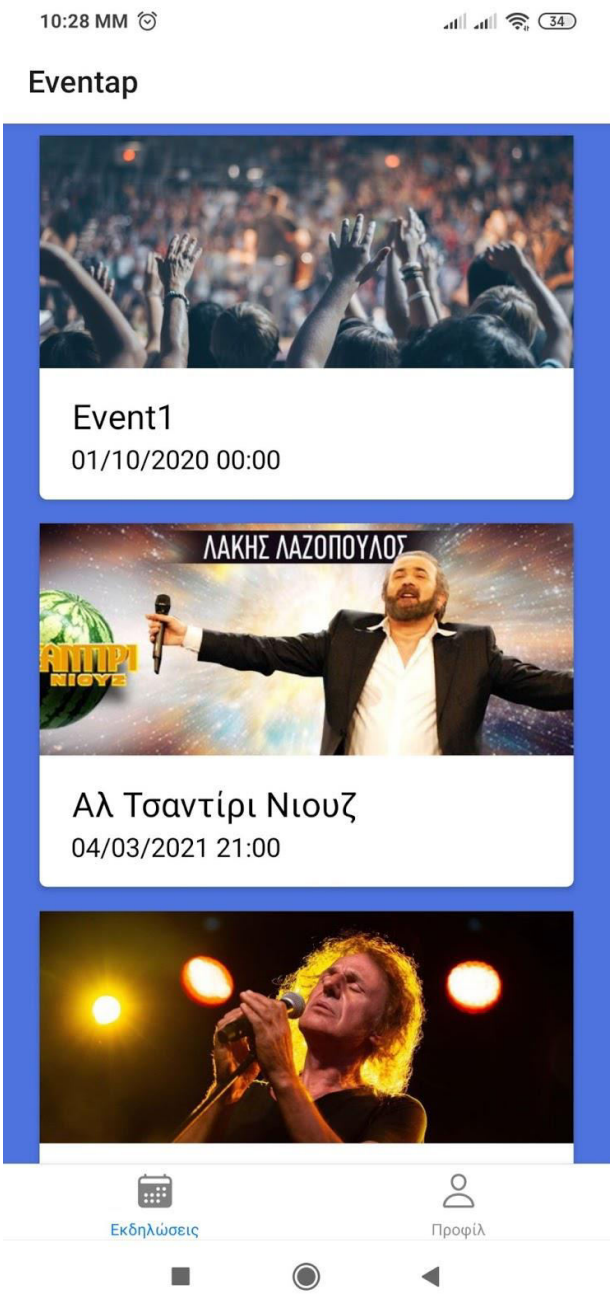
Στην παρούσα ενότητα παρουσιάζονται στιγμιότυπα από τις οθόνες της εφαρμογής. Δόθηκε ιδιαίτερη έμφαση ώστε το UI να είναι απλό και εύχρηστο. Καταρχάς υλοποιήθηκαν σχεδιαστικά σε ειδική εφαρμογή δημιουργίας mockup, την figma. [8]

Παρακάτω ακολουθούν τα στιγμιότυπα.



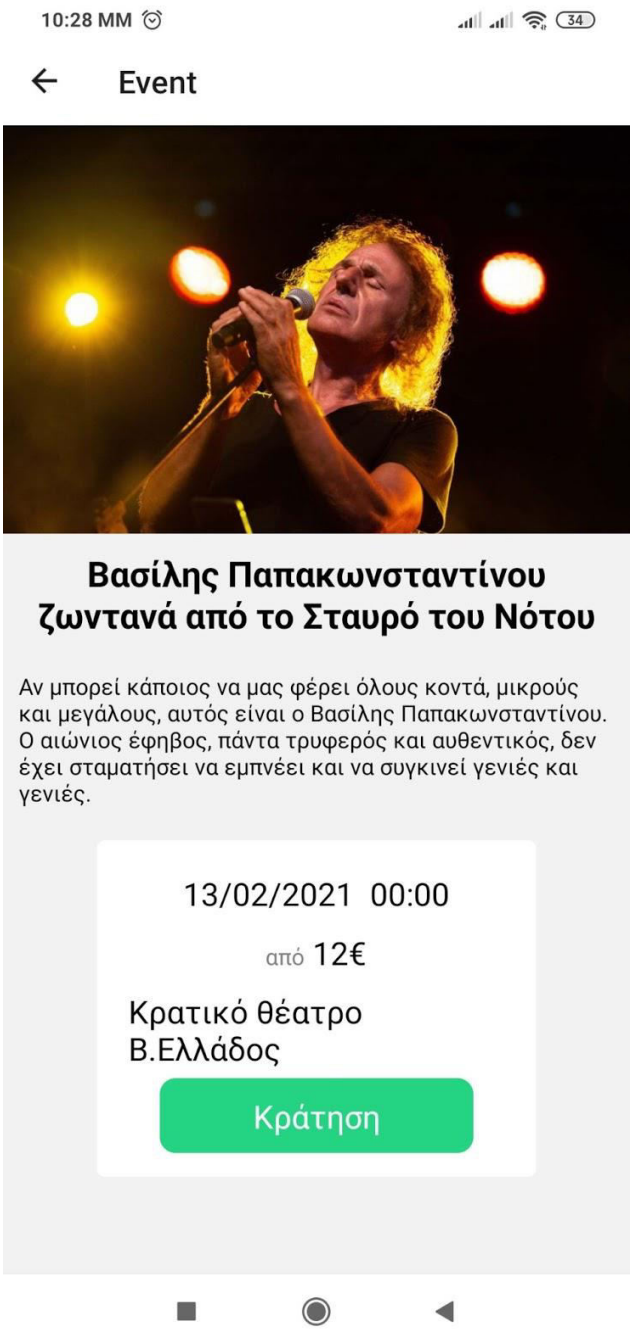
Εικόνα 1: Στιγμιότυπο της οθόνης σύνδεσης της εφαρμογής για κινητά

Η παραπάνω οθόνη είναι η οθόνη σύνδεσης στην εφαρμογή και κατ επέκταση η πρώτη οθόνη που βλέπει κάποιος χρήστης. Υπάρχουν δύο πεδία για να εισάγει το email και τον κωδικό του και το κουμπί σύνδεση που όταν το πατήσει εκτελείται ο κώδικας ελέγχου στοιχείων. Υπάρχουν άλλα δύο κουμπιά, ένα αν ο χρήστης ξεχάσει τον κωδικό του και στο κάτω μέρος της οθόνης ένα ακόμη σε περίπτωση που χρειάζεται γενική βοήθεια με την εφαρμογή. Και τα δύο παραπέμπουν σε διαφορετικά alerts που ενημερώνουν τον χρήστη που πρέπει να απευθυνθεί. Εφόσον ο χρήστης συνδεθεί τότε του εμφανίζεται η παρακάτω οθόνη.



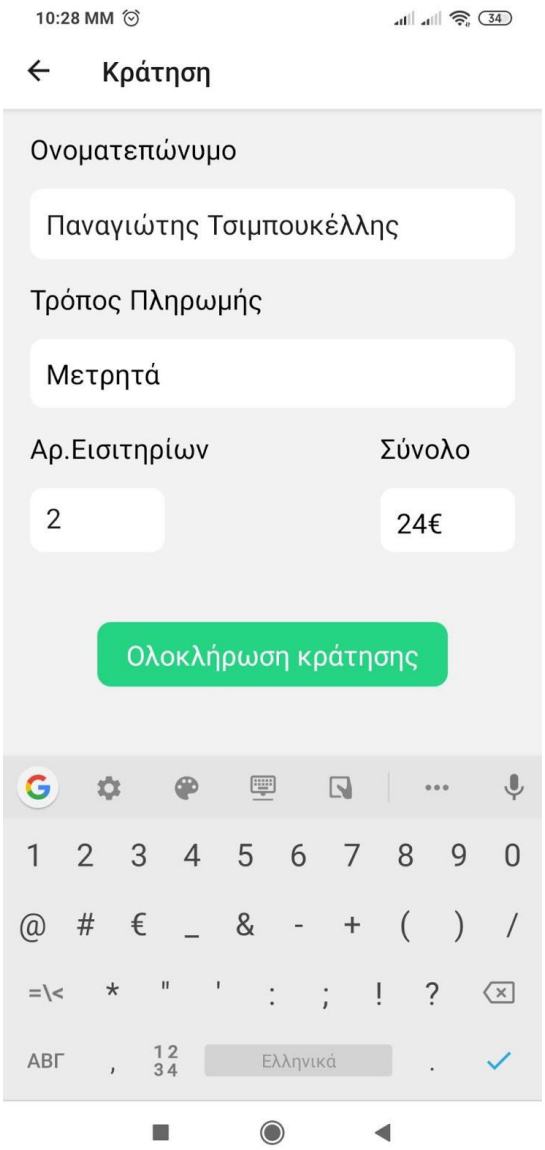
Εικόνα 2: Στιγμιότυπο της οθόνης των εκδηλώσεων της εφαρμογής για κινητά

Στην παραπάνω οθόνη ο χρήστης βλέπει τις εκδηλώσεις που υπάρχουν διαθέσιμες και έχει τη δυνατότητα να επιλέξει κάποια ώστε να δει περισσότερες πληροφορίες και να κάνει κράτηση. Στο κάτω μέρος της συγκεκριμένης οθόνης υπάρχει το μενού από το οποίο μπορεί να πηγαίνει στο προφιλ του ή στις εκδηλώσεις. Σε περίπτωση που επιλέξει κάποια εκδήλωση, θα μεταφερθεί στην παρακάτω οθόνη.



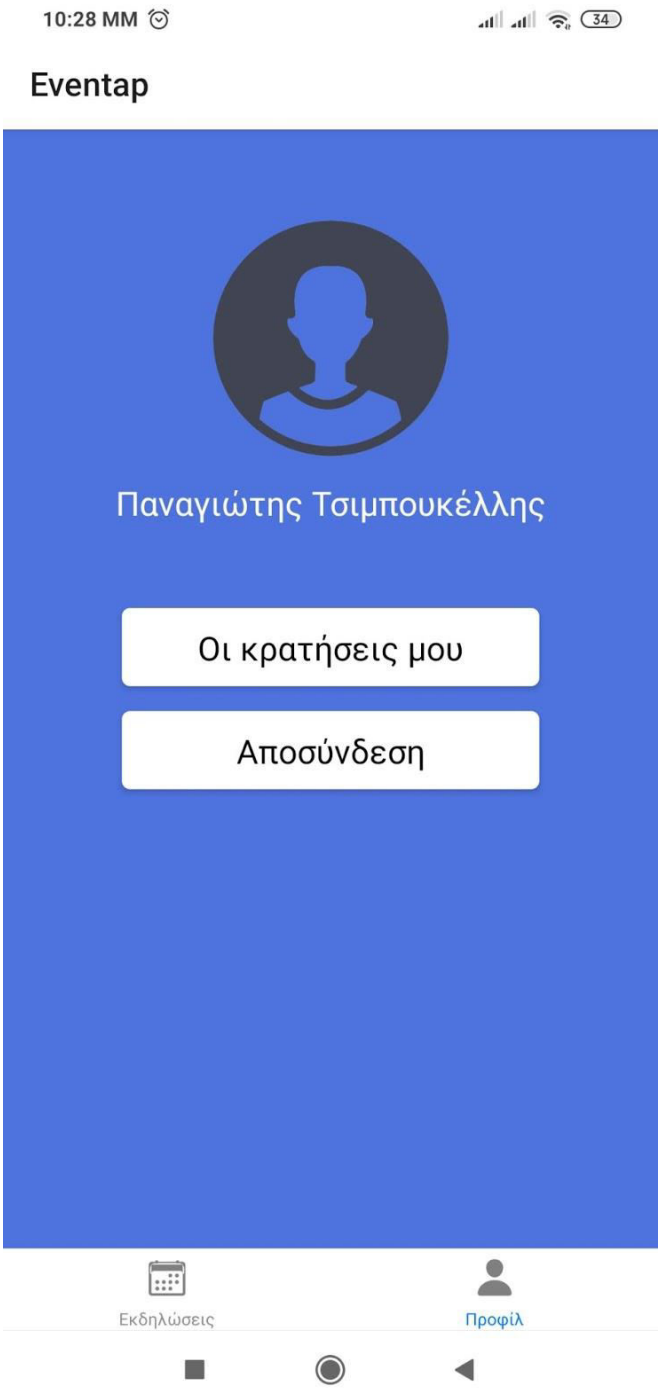
Εικόνα 3: Στιγμιότυπο της οθόνης μιας συγκεκριμένης εκδήλωσης της εφαρμογής για κινητά

Στη συγκεκριμένη οθόνη ο χρήστης μπορεί να δει περισσότερες πληροφορίες για την εκδήλωση που επέλεξε, όπως για παράδειγμα την περιγραφή της και τις διαθέσιμες ώρες και ημερομηνίες αλλά και την τιμή του εισιτηρίου. Σε αυτή την οθόνη οι επιλογές που έχει είναι δύο. Είτε να πάει πίσω από το βελάκι στο πάνω αριστερά μέρος της οθόνης είτε να κάνει κράτηση για την συγκεκριμένη εκδήλωση. Σε περίπτωση που επιλέξει να κάνει κράτηση βλέπει την παρακάτω οθόνη.



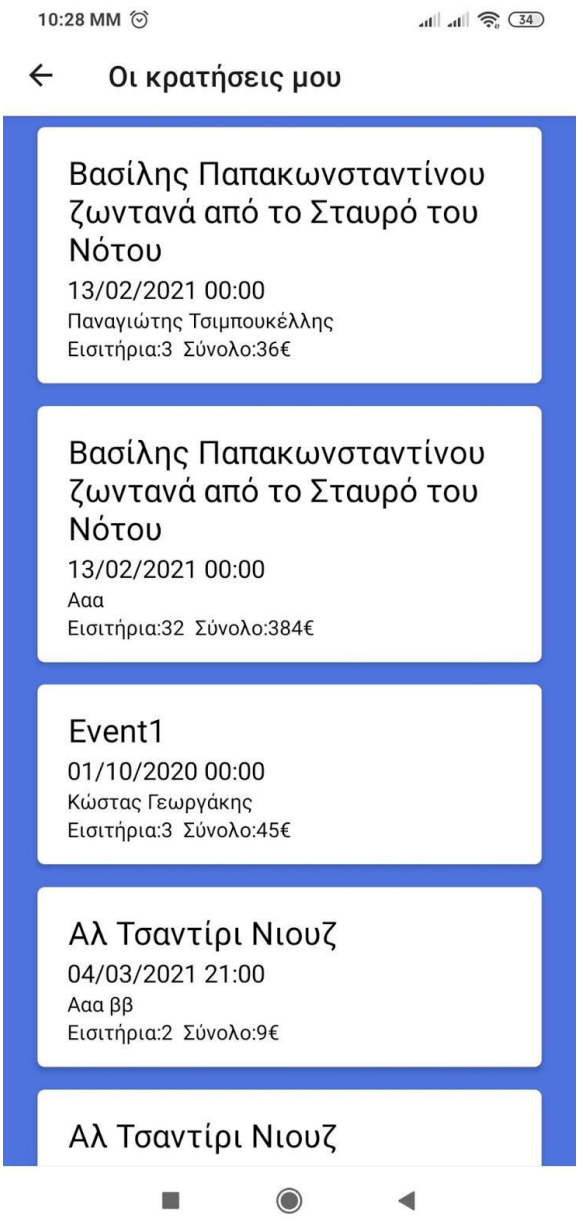
Εικόνα 4:Στιγμιότυπο από την οθόνη κράτησης μιας εκδήλωσης της εφαρμογής για κινητά

Στη συγκεκριμένη οθόνη ο χρήστης συμπληρώνοντας τα στοιχεία του μπορεί να ολοκληρώσει την κράτηση για μια συγκεκριμένη εκδήλωση. Οι πληροφορίες που μπορεί να συμπληρώσει είναι το ονοματεπώνυμο στο οποίο θα γίνει η κράτηση και τον αριθμό των εισιτηρίων. Έτσι υπολογίζεται το συνολικό ποσό που πρέπει να καταβάλλει όταν φτάσει στο ταμείο της εκδήλωσης.



Εικόνα 5: Στιγμιότυπο της οθόνης του προφίλ ενός χρήστη της εφαρμογής για κινητά

Η παραπάνω οθόνη εμφανίζεται όταν ο χρήστης βρίσκεται στις εκδηλώσεις και στο κάτω μενού επιλέξει να δει το προφιλ του. Από εκεί μπορεί να δει τις κρατήσεις που έχει ή να κάνει αποσύνδεση. Σε περίπτωση που επιλέξει να δει τις κρατήσεις που έχει, του εμφανίζεται η παρακάτω οθόνη.

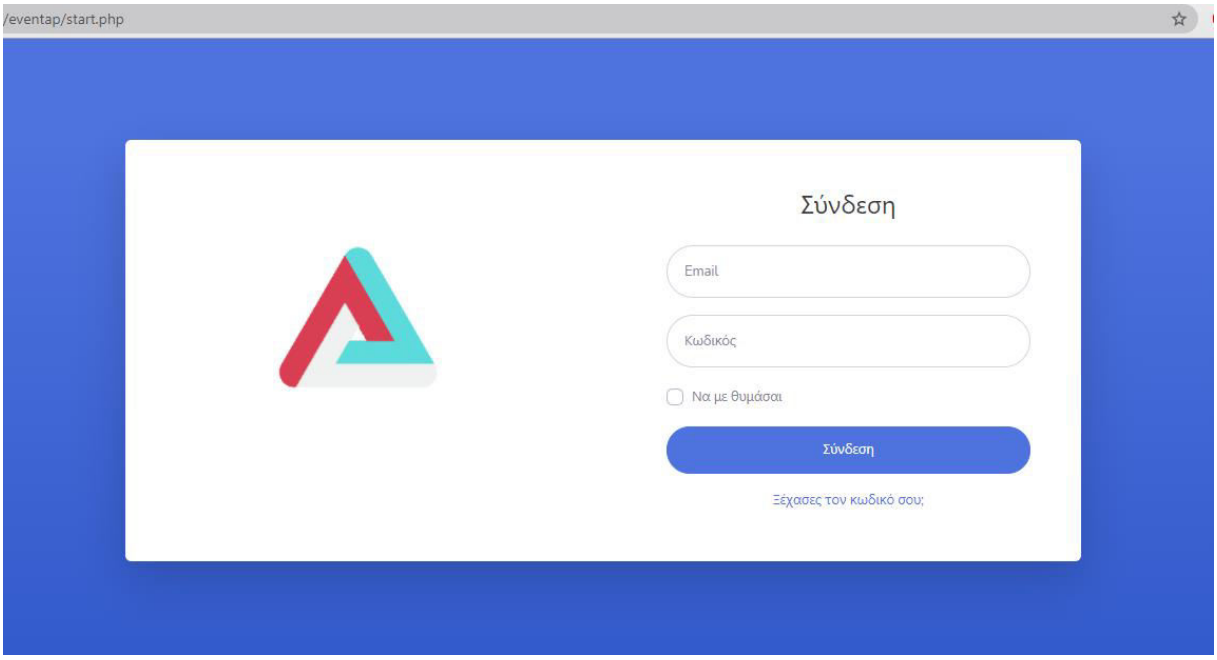


Εικόνα 6: Στιγμιότυπο της οθόνης των κρατήσεων ενός χρήστη της εφαρμογής για κινητά

Στη συγκεκριμένη οθόνη ο χρήστης μπορεί να δει τις κρατήσεις που έχει κάνει. Αναλυτικότερα βλέπει τον τίτλο της εκδήλωσης, την ημερομηνία, το ονοματεπώνυμο στο οποίο έκανε την κράτηση, τον αριθμό των εισιτηρίων που επέλεξε και το συνολικό κόστος

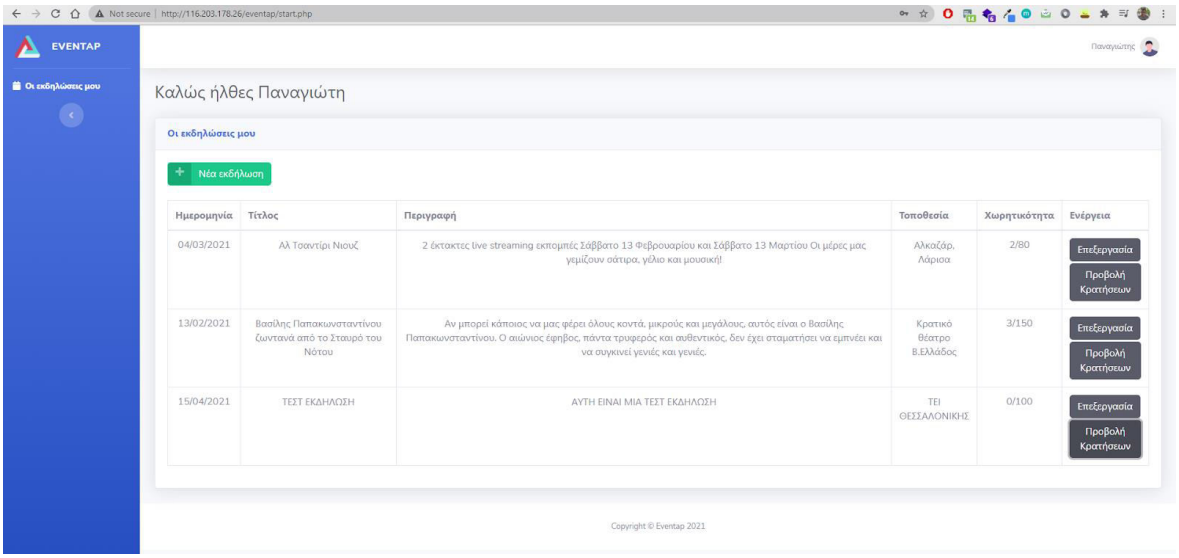
2.3 Wireframes Web App

Στη συγκεκριμένη ενότητα παρουσιάζονται οι οθόνες της εφαρμογής για διοργανωτές εκδηλώσεων. Για τη συγκεκριμένη web εφαρμογή, χρησιμοποιήθηκε ένα δωρεάν template (https://startbootstrap.com/) . Ο σκοπός χρήσης του συγκεκριμένου template είναι να μπορεί να υποστηρίξει μια ολοκληρωμένη διαχειριστική πλατφόρμα εκδηλώσεων.



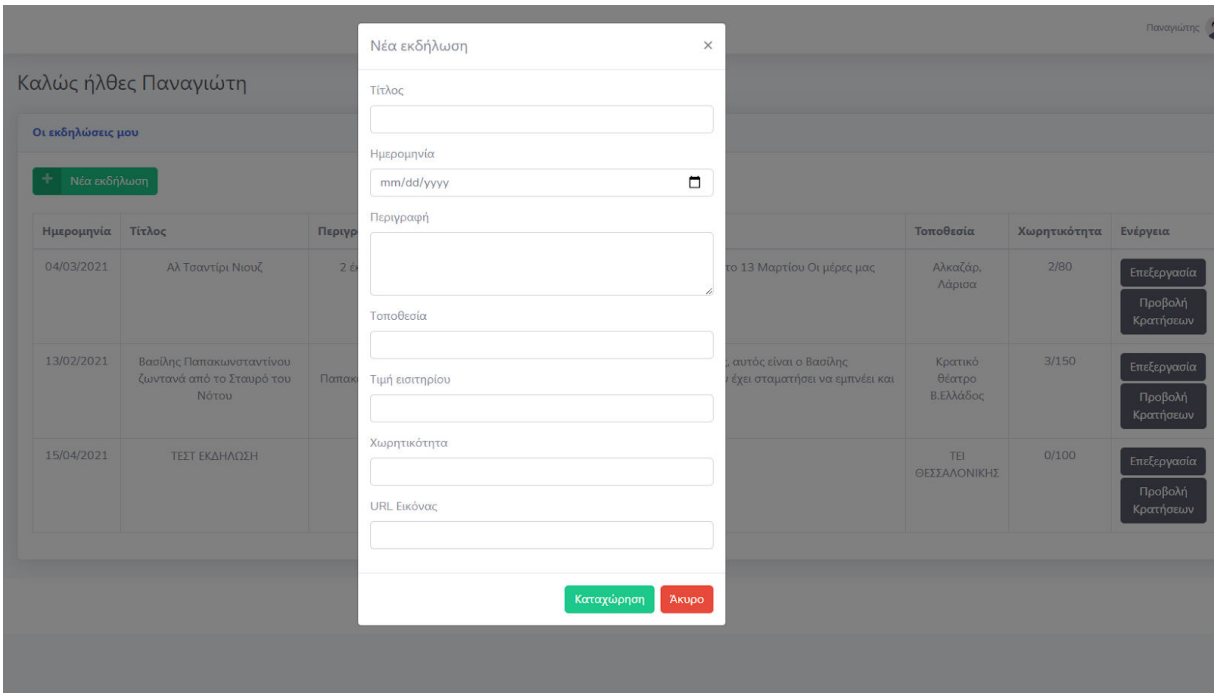
Εικόνα 7: Στιγμιότυπο οθόνης σύνδεσης στην web εφαρμογή για διοργανωτές εκδηλώσεων

Η παραπάνω οθόνη είναι αυτή την οποία βλέπει κάποιος διαχειριστής εκδηλώσεων που έχει πληκτρολογήσει το url από τον υπολογιστή του. Όπως και ο χρήστης στην εφαρμογή για κινητά, έτσι και ο διαχειριστής εκδηλώσεων είναι υποχρεωτικό να συμπληρώσει το email και τον κωδικό του. Υπάρχει η δυνατότητα επιλέγοντας το "Να με θυμάσαι" να αποθηκευτούν τοπικά τα στοιχεία του χρήστη έτσι ώστε από την επόμενη φορά να πατάει απλώς σύνδεση. Επιπλέον σε περίπτωση που ξεχάσει τον κωδικό του και επιλέξει το "Ξέχασες τον κωδικό σου;" εμφανίζεται αντίστοιχο μήνυμα που τον ενημερώνει για τα βήματα που πρέπει να κάνει. Από τη στιγμή που ο διαχειριστής εκδηλώσεων συμπληρώσει σωστά τα στοιχεία του και γίνει επιτυχώς η σύνδεση μεταφέρεται στην παρακάτω οθόνη.



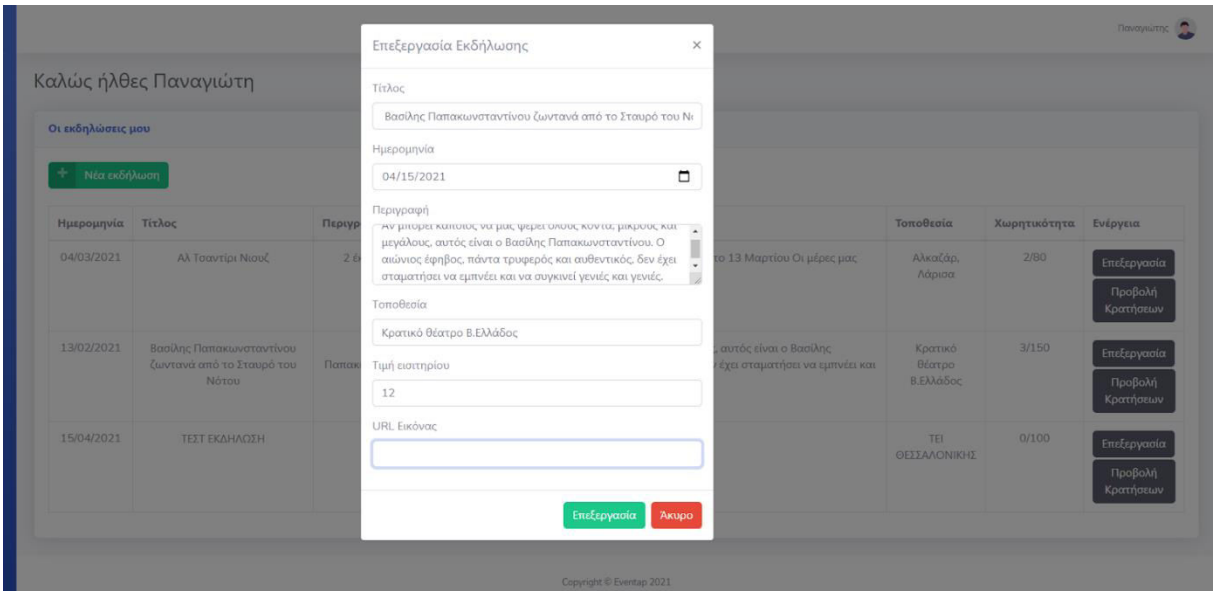
Εικόνα 8: Στιγμιότυπο κεντρικής οθόνης στην web εφαρμογή για διοργανωτές εκδηλώσεων

Στη συγκεκριμένη οθόνη ο διαχειριστής βλέπει τη λίστα με τις δικές του εκδηλώσεις. Μπορεί να δημιουργήσει νέα εκδήλωση, να επεξεργαστεί κάποια ήδη υπάρχουσα αλλά και να δει τις κρατήσεις για κάποια ήδη υπάρχουσα. Σε περίπτωση που θέλει να δημιουργήσει νέα εκδήλωση πατάει το κουμπί "Νέα Εκδήλωση" και εμφανίζεται το modal της παρακάτω οθόνης ώστε να εισάγει τα στοιχεία της



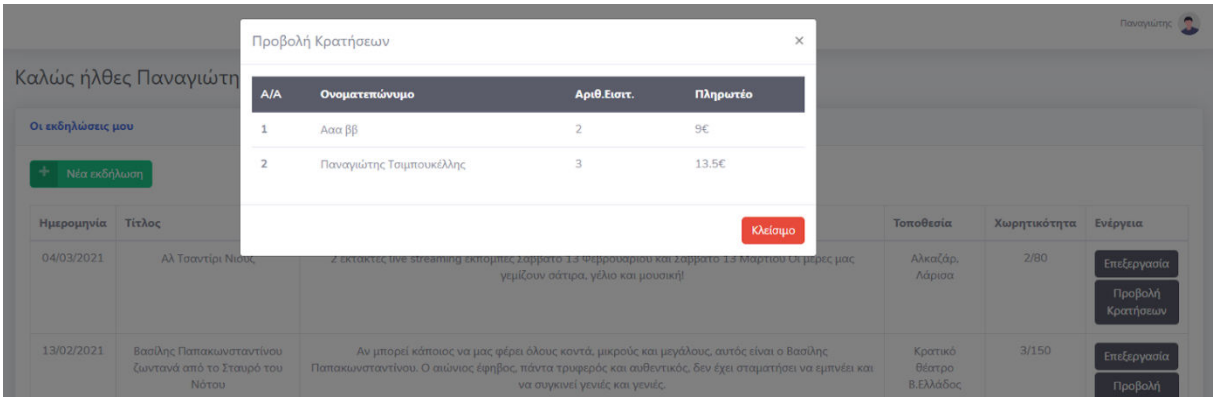
Εικόνα 9: Στιγμιότυπο οθόνης δημιουργίας νέας εκδήλωσης στην εφαρμογή για διαχείριση εκδηλώσεων

Στην παραπάνω οθόνη ο διαχειριστής μπορεί να συμπληρώσει όλα τα απαραίτητα στοιχεία ώστε να δημιουργήσει την εκδήλωση. Τα στοιχεία είναι: τίτλος, ημερομηνία, περιγραφή, τοποθεσία, τιμή εισιτηρίου, χωρητικότητα και url εικόνας της εκδήλωσης. Στη συνέχεια κάνοντας καταχώρηση η εκδήλωση δημιουργείται και εμφανίζεται στην εφαρμογή για κινητά. Σε περίπτωση που ο διαχειριστής θέλει να επεξεργαστεί μια ήδη υπάρχουσα εκδήλωση κάνει κλικ στο κουμπί επεξεργασία στο δεξί μέρος της οθόνης και εμφανίζεται το παρακάτω modal.



Εικόνα 10: Στιγμιότυπο οθόνης επεξεργασίας υπάρχουσας εκδήλωσης στην εφαρμογή για διαχείριση εκδηλώσεων

Στο παραπάνω modal ο διαχειριστής μπορεί να επεξεργαστεί κάποιο ή κάποια από τα στοιχεία της εκδήλωσης



Εικόνα 11: Στιγμιότυπο της οθόνης προβολής κρατήσεων μιας εκδήλωσης της εφαρμογής διαχείρισης εκδηλώσεων

Τέλος αν θέλει να δει τις κρατήσεις που υπάρχουν σε μια εκδήλωση πατώντας το κουμπί “Προβολή Κρατήσεων” εμφανίζεται το αντίστοιχο modal το οποίο περιέχει ένα table που έχει ανά γραμμή την κάθε κράτηση.

2.4 Επίλογος

Σε αυτό το σημείο έχουμε τελειώσει με την εισαγωγή της εφαρμογής, την γενική εικόνα και το User Interface. Ήρθε λοιπόν η στιγμή της ανάλυσης τεχνικών θεμάτων και η δημιουργία των δύο servers, όπου ο ένας θα στεγάσει το API και ο άλλος, αυτό που είδαμε μόλις, δηλαδή τη web εφαρμογή διαχείρισης εκδηλώσεων.

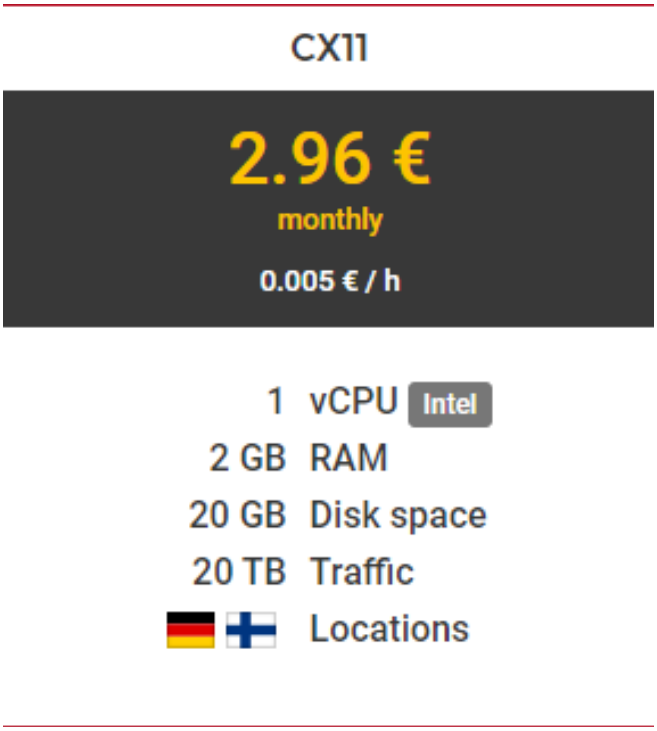
Κεφάλαιο 3ο: Δημιουργία servers

3.1 Εισαγωγή

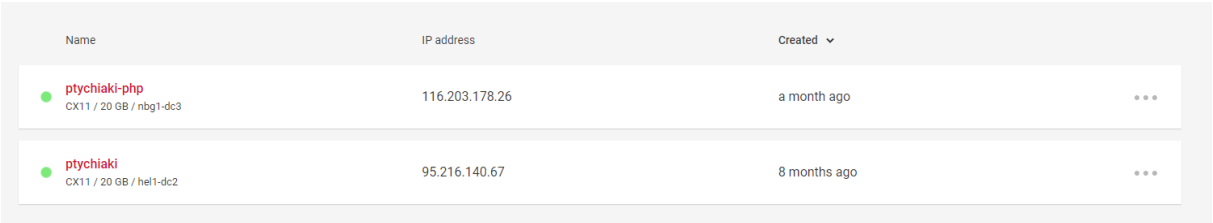
Στην παρούσα ενότητα θα αναλύσουμε τον τρόπο απόκτησης αλλά και παραμετροποίησης ενός VPS, δηλαδή ενός εικονικού server σε λειτουργικό σύστημα Ubuntu[11]. Το Ubuntu είναι το πλέον δημοφιλέστερο λογισμικό για τέτοιου είδους εφαρμογές. Τέτοιου είδους servers φιλοξενούνται σε φυσικούς servers σε μεγάλα data centers. Ένας φυσικός server μπορεί να φιλοξενήσει από έναν μέχρι εκατοντάδες VPS.

3.2 Ενοικίαση και παραμετροποίηση server

Για τις ανάγκες της πτυχιακής εργασίας, προχώρησα στην ενοικίαση 2 VPS servers από την Hetzner. Η Hetzner είναι web hosting provider με έδρα τη Γερμανία. Οι 2 servers είναι πανομοιότυποι με τα παρακάτω χαρακτηριστικά:



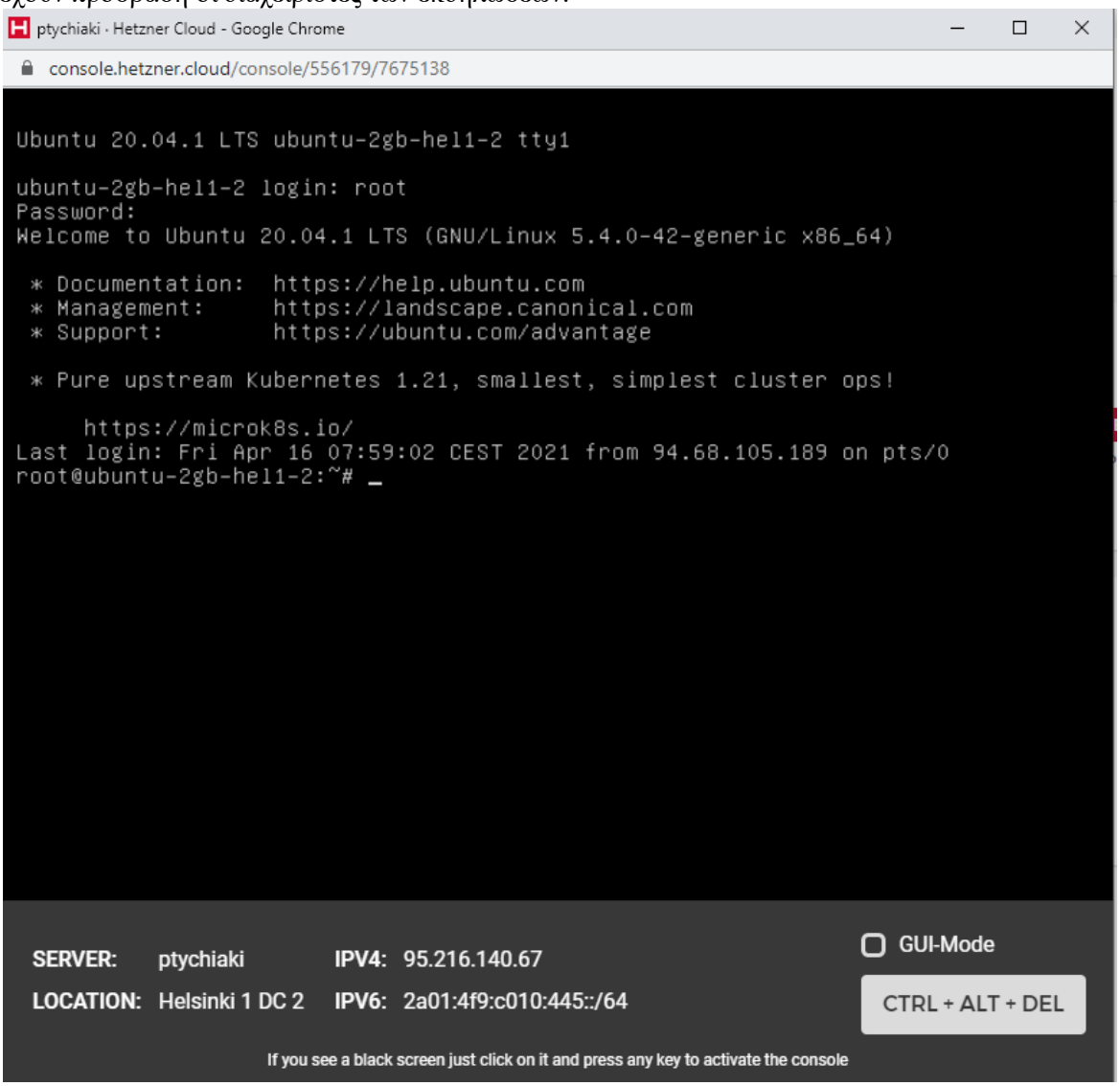
Εικόνα 12: Χαρακτηριστικά του server (CX11)



Εικόνα 13: Λίστα με τους servers

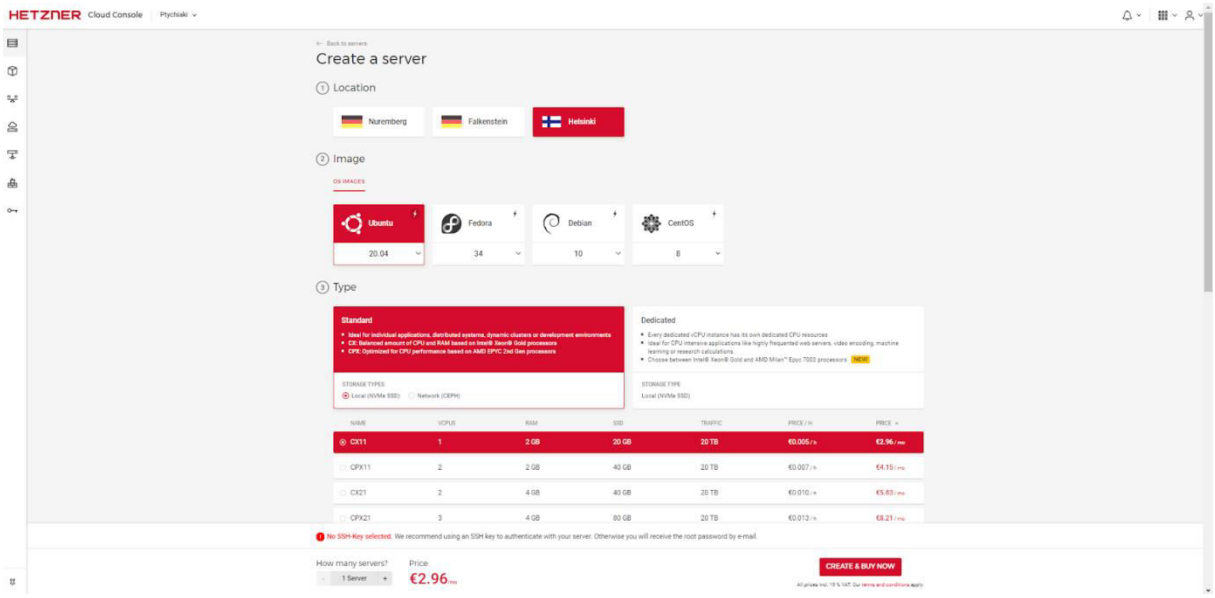
Και στους 2 servers εγκατέστησα Ubuntu 20.04 και Apache. Στον έναν από τους δύο εγκατέστησα node.js ώστε να κατασκευάσω πάνω του το REST Api Service[9] το οποίο θα καλεί και η mobile αλλά και η web εφαρμογή για να μπορούν να διεκπεραιώνονται τα ερωτήματα στη βάση δεδομένων.

Στον δεύτερο εγκατέστησα PHP ώστε να κατασκευάσω πάνω του τη Web εφαρμογή στην οποία θα έχουν πρόσβαση οι διαχειριστές των εκδηλώσεων.

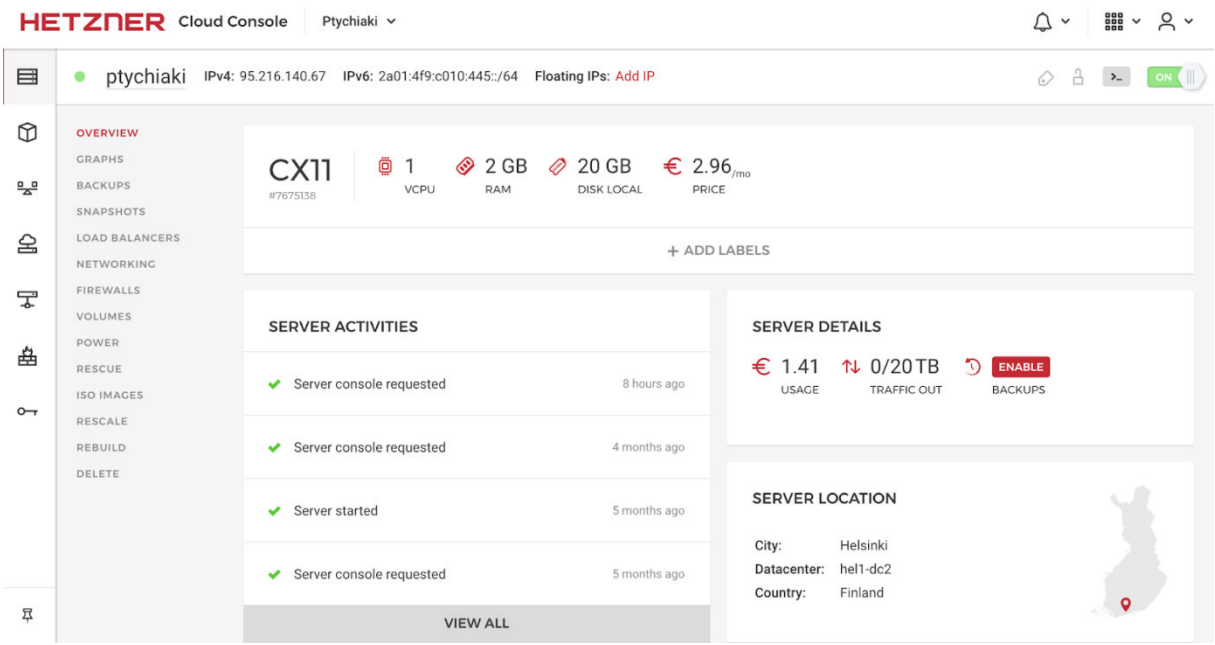


Εικόνα 14: Η κονσόλα που δίνει η Hetzner

Για να επιλέξω σε ποια εταιρεία θα στεγάσω το project έκανα έρευνα σε διάφορα websites αλλά και ομάδες προγραμματιστών, ώστε να μάθω σε ποιους παρόχους καταφεύγουν επαγγελματίες προγραμματιστές για έργα πελατών. Πολύ σημαντικό είναι ότι το interface το οποίο έχει είναι πολύ φιλικό προς τον χρήστη και μπορείς εύκολα και γρήγορα να δημιουργήσεις έναν server. Ένα ακόμη πολύ σημαντικό είναι ότι με ένα click μπορείς να ανοίξεις την κονσόλα του server, η οποία ανοίγει σε ένα νέο παράθυρο στον browser. Διαφορετικά μπορείς να συνδεθείς από κάποιο άλλο λογισμικό (πχ putty).



Εικόνα 15: Περιβάλλον δημιουργίας server στη Hetzner



Εικόνα 16: Περιβάλλον προβολής ενός server

3.3 Εγκατάσταση στον πρώτο server (node.JS - API)

Το πρώτο βήμα που πρέπει να κάνουμε αφού έχουμε δημιουργήσει τον server είναι να εγκαταστήσουμε node.js ώστε να καταφέρουμε να δημιουργήσουμε το RESTful API. Το node.js είναι μια πλατφόρμα ανάπτυξης λογισμικού για servers, σε γλώσσα javascript. Παρακάτω είναι οι εντολές που χρειάζονται για την εγκατάσταση του node.js:

```
sudo apt update
```

Η συγκεκριμένη εντολή ελέγχει αν όλα τα πακέτα που είναι εγκατεστημένα στον server είναι ενημερωμένα. Αν κάποιο δεν είναι, τότε το ενημερώνει.

```
sudo apt install nodejs
```

Η συγκεκριμένη εντολή εγκαθιστά την node.js

Εφόσον η εγκατάσταση έχει γίνει σωστά τότε εκτελώντας την παρακάτω εντολή λαμβάνουμε ως απάντηση την έκδοση του node.js

```
nodejs -v
```

Καθώς θα χρειαστούμε κάποια επιπλέον πακέτα για την ομαλή και σωστή ανάπτυξη του API είναι αναγκαίο να εγκαταστήσουμε το node package manager εκτελώντας την παρακάτω εντολή

```
sudo apt install npm
```

Στη συνέχεια πρέπει να εγκαταστήσουμε τον Apache ο οποίος θα κάνει τον server να λειτουργεί ως Web Server και να δέχεται κλήσεις (GET , POST etc). Η εντολή που πρέπει να εκτελέσουμε για να εγκατασταθεί είναι η παρακάτω:

```
sudo apt install apache2
```

Εφόσον όλα τα παραπάνω εκτελεστούν επιτυχώς είμαστε έτοιμοι για να προχωρήσουμε στην ανάπτυξη του API.

3.4 Εγκατάσταση στον δεύτερο server (PHP – Web App)

Σε αυτόν τον server θα κάνουμε τα ίδια βήματα όσον αφορά την εγκατάσταση του Apache, όμως στη συνέχεια θα εγκαταστήσουμε την PHP με την παρακάτω εντολή:

```
sudo apt install php libapache2-mod-php
```

Στη συνέχεια και εφόσον έχει εκτελεστεί με επιτυχία η παραπάνω εντολή τότε μπορούμε να εγκαταστήσουμε και το fpm (Fast Process Manager), το οποίο είναι ένα web εργαλείο που είναι υπεύθυνο για την μεγιστοποίηση της ταχύτητας και της απόδοσης ενός web server. Για να το εγκαταστήσουμε εκτελούμε την παρακάτω εντολή:

```
sudo apt install php-fpm
```

Εφόσον κι αυτή η εντολή έχει εκτελεστεί επιτυχώς, τέλος θα πρέπει να επανεκκινήσουμε τον Apache με την παρακάτω εντολή ώστε να προχωρήσουμε στη δημιουργία της web εφαρμογής

```
sudo systemctl restart apache2
```

3.5 Επίλογος

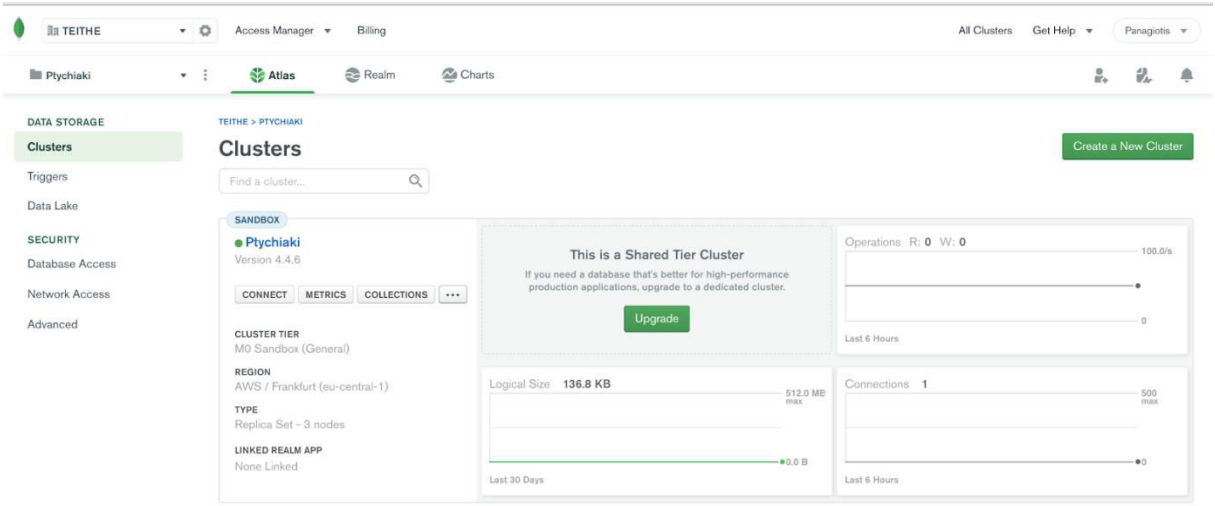
Παραπάνω αναλύσαμε την παραμετροποίηση που χρειάζεται ένα VPS ώστε να φιλοξενήσει μια εφαρμογή. Τα τελευταία χρόνια υπάρχουν έτοιμα εργαλεία που απλοποιούν αρκετά τέτοιου είδους παραμετροποιήσεις ώστε ο χρήστης με ελάχιστες εντολές να εκτελέσει διεργασίες που στο παρελθόν απαιτούσαν πολύ περισσότερο χρόνο και κόπο. Αξίζει να αναφερθεί πως πέρα από τον τρόπο που χρησιμοποίησαμε παραπάνω, υπάρχουν και άλλες λύσεις για τις συγκεκριμένες εργασίες, που εκεί δεν χρειάζεται ο προγραμματιστής να εκτελέσει κάποια εντολή. Παρακάτω θα αναλύσουμε τον τρόπο δημιουργίας της βάσης δεδομένων

Κεφάλαιο 4ο:MongoDB

4.1 Εισαγωγή

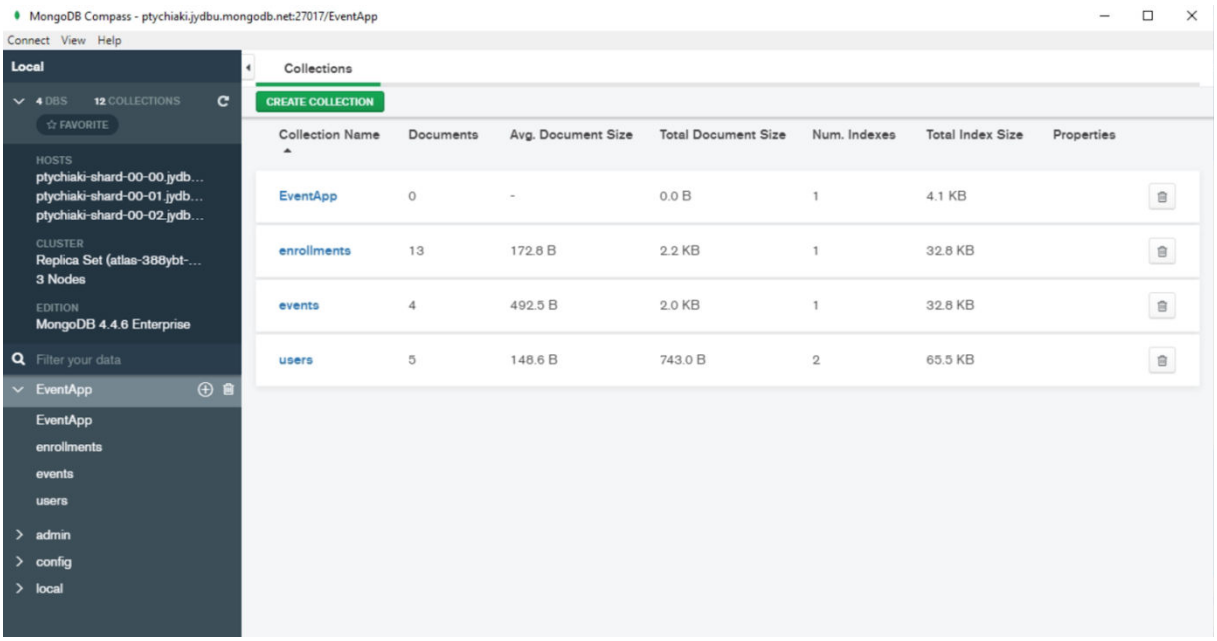
Η MongoDB είναι μια μη σχεσιακή (NoSQL) document oriented βάση δεδομένων, η οποία χρησιμοποιείται κατά κύριο λόγο σε εφαρμογές που χρειάζεται να αποθηκεύουν πάρα πολλά δεδομένα. Σε αντίθεση με μια SQL βάση δεδομένων η οποία έχει πίνακες και γραμμές, η MongoDB χρησιμοποιεί τις συλλογές και τα έγγραφα (collections & documents). Τα έγγραφα (documents) αποτελούνται από ζευγάρια key-value. Αυτή είναι και η βασική δομή της MongoDB.

Ήδη χρησιμοποιείται σε εφαρμογές εταιρειών κολοσσών, όπως η Google, η Adobe, η SAP, το Ebay και η Sega. Πέρα από τη βάση δεδομένων η εταιρεία προσφέρει διάφορα εργαλεία ώστε οι προγραμματιστές να κάνουν εύκολα και γρήγορα τη διασύνδεση με την εφαρμογή τους. Ένα από αυτά τα εργαλεία είναι το Atlas Cloud, το οποίο προσφέρεται είτε δωρεάν είτε με συνδρομή (αναλόγως τον όγκο των δεδομένων) και επιτρέπει σε έναν προγραμματιστή να διαχειρίζεται τη βάση δεδομένων του, την οποία στεγάζει στο cloud, από ένα έτοιμο web interface που παρέχει η εταιρεία.



Εικόνα 17: : Στιγμιότυπο οθόνης web interface της MongoDB (Atlas)

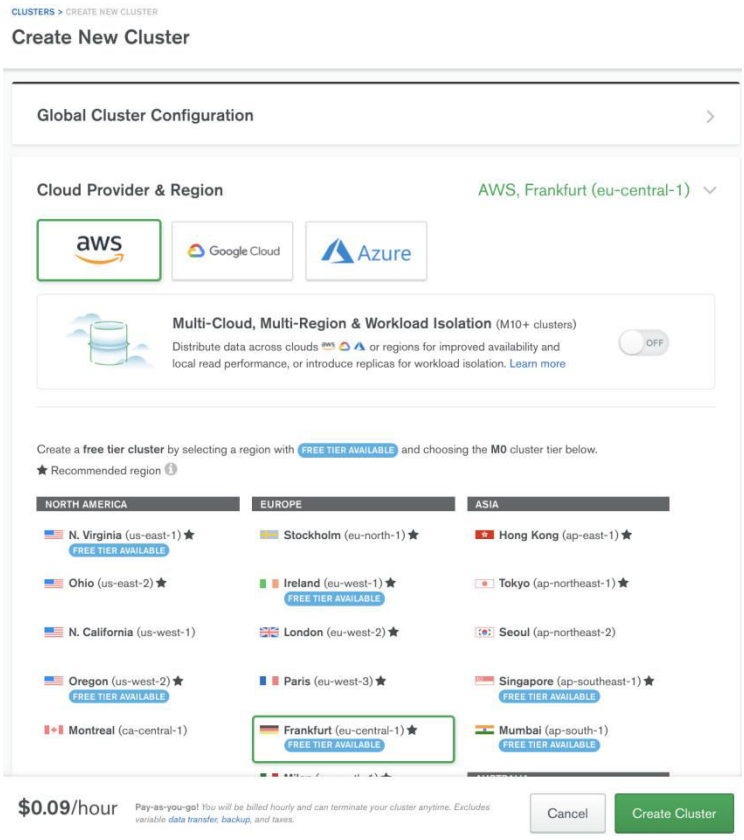
Φυσικά πέρα από τις cloud λύσεις, υπάρχει και η δυνατότητα εγκατάστασης μιας desktop εφαρμογής, η οποία και αυτή είναι προϊόν της Mongo. Με αυτή τη desktop εφαρμογή, ο προγραμματιστής μπορεί να κάνει σχεδόν όλες τις εργασίες που κάνει και στην Cloud λύση.



Εικόνα 18: Στιγμιότυπο οθόνης web interface της MongoDB (Atlas)

4.2 Δημιουργία Βάσης Δεδομένων

Μετά την δωρεάν εγγραφή, κάνοντας ένα κλικ, μεταφέρεσαι σε μια σελίδα για τη δημιουργία της βάσης δεδομένων.



Εικόνα 19: οθόνης της σελίδας δημιουργίας Βάσης Δεδομένων

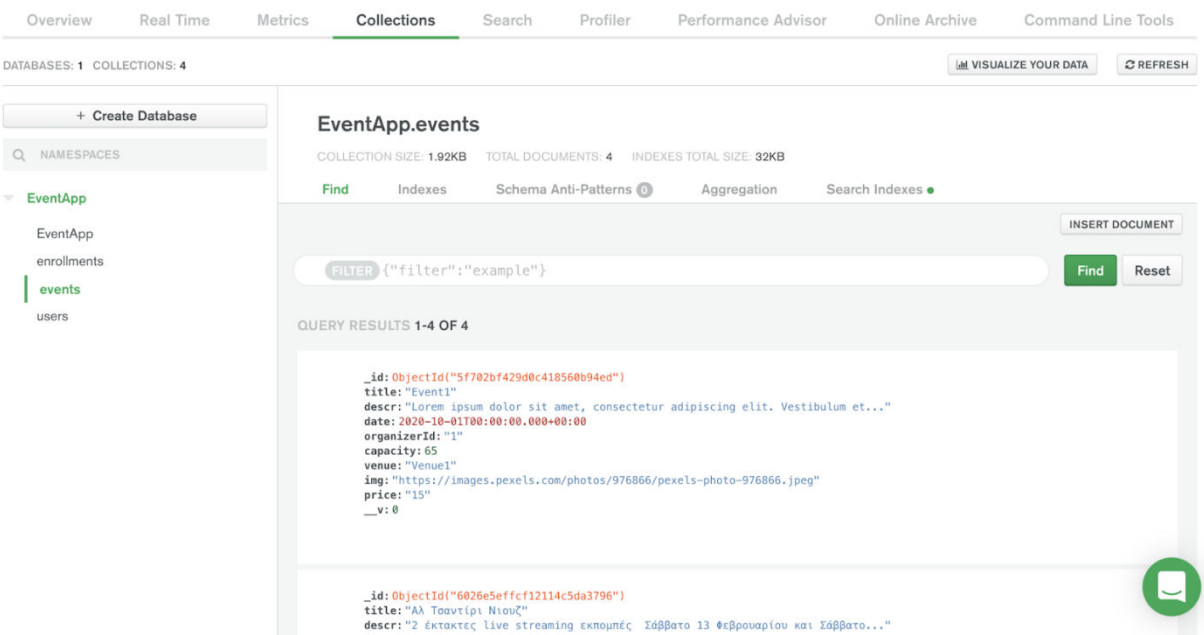
Αυτό το οποίο πρέπει να προσέξουμε κατά τη δημιουργία του cluster είναι να βρίσκεται όσο πιο κοντά γίνεται στην Ελλάδα, εκεί δηλαδή όπου θα χρησιμοποιείτε η εφαρμογή, ώστε να υπάρχει όσο το δυνατόν μικρότερο ping (καθυστέρηση). Με αυτό τον τρόπο οι χρήστες θα απολαμβάνουν μια καλύτερη εμπειρία, καθώς οι κλήσεις από και προς τη βάση δεδομένων θα διεκπαιρεώνονται σε μικρότερο χρονικό διάστημα, σχεδόν ακαριαία και άρα δεν θα υπάρχει κάποια αντιληπτή καθυστέρηση.

Επέλεξα να στεγάσω τη συγκεκριμένη βάση στο data center της Φρανκφούρτης και συγκεκριμένα στη δωρεάν έκδοση, η οποία προσφέρει 512MB συνολικό χώρο. Ένα πολύ μεγάλο πλεονέκτημα είναι ότι μετά τη δημιουργία της είναι έτοιμη για χρήση μέσα σε ελάχιστα λεπτά.

4.3 Δομή της βάσης δεδομένων

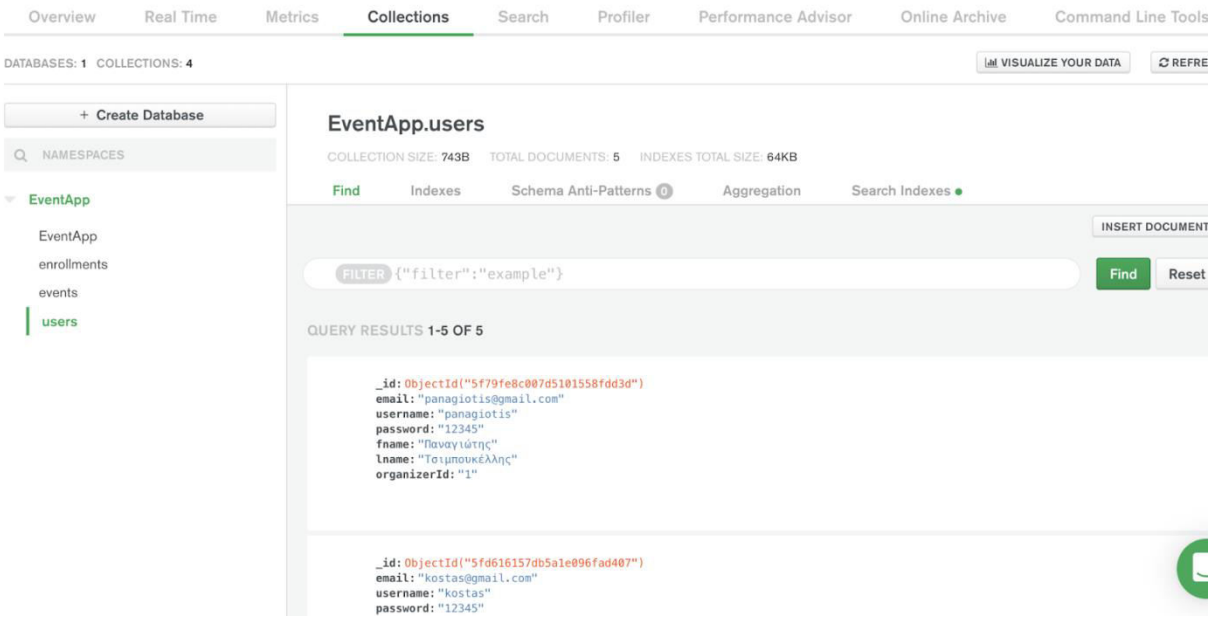
Η βάση δεδομένων αποτελείται από τρία βασικά collections. Αυτά είναι τα events, οι users και τα enrollments. Παρακάτω αναλύουμε το καθένα:

- **events:** Τα events είναι η κύρια συλλογή της εφαρμογής. Τα έγγραφα που περιέχονται στη συγκεκριμένη συλλογή αναπαριστούν τις εκδηλώσεις οι οποίες δημιουργούνται στη web εφαρμογή από τους διοργανωτές και προβάλλονται στην mobile εφαρμογή από τους χρήστες. Τα πεδία που έχει το κάθε έγγραφο είναι τα παρακάτω:
    - **_id**: Μοναδικό κλειδί (αναφέρεται ως ObjectId) και δημιουργείται αυτόματα από τη Mongo
    - **title**: Ο τίτλος της εκδήλωσης
    - **descr**: Η περιγραφή της εκδήλωσης
    - **date**: Η ημερομηνία διεξαγωγής της εκδήλωσης
    - **organizerId**: Το id του διοργανωτή της εκδήλωσης
    - **capacity**: Η χωρητικότητα της συγκεκριμένης εκδήλωσης
    - **venue**: Η τοποθεσία της εκδήλωσης
    - **img**: Το url με την εικόνα της εκδήλωσης, με σκοπό την προβολή της στην εφαρμογή για κινητά.
    - **price**: Η τιμή του εισιτηρίου



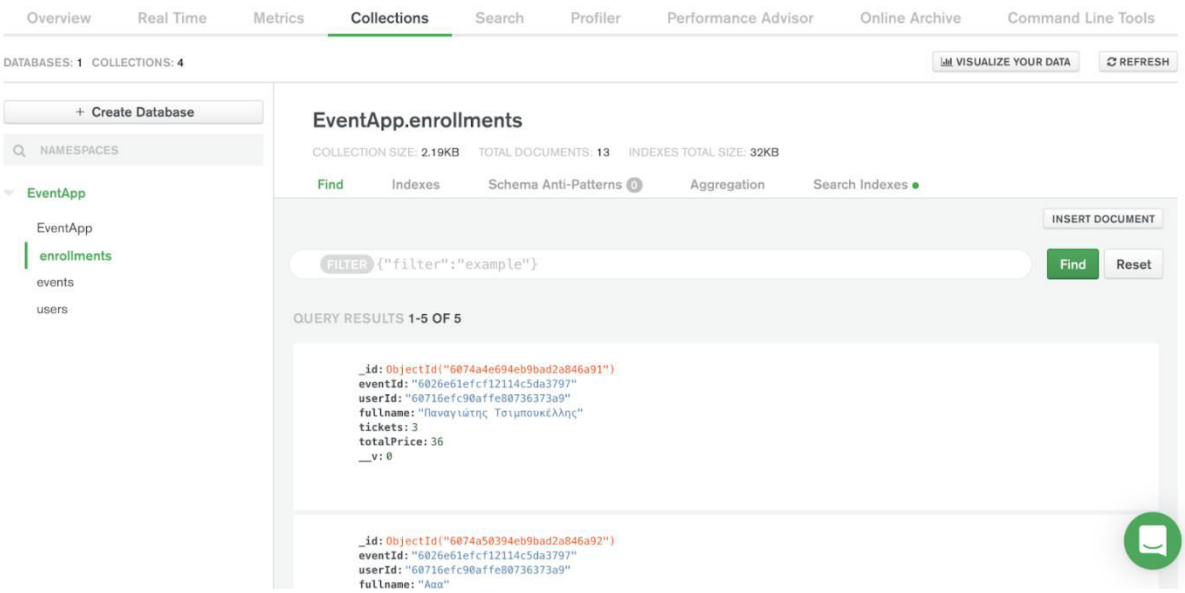
Εικόνα 20: Στιγμιότυπο του collection των events

- **users**: Σε αυτή τη συλλογή αποθηκεύονται όλοι οι χρήστες της εφαρμογής, ανεξάρτητα από το αν είναι διοργανωτές ή απλοί χρήστες που απλώς θέλουν να κάνουν μια κράτηση για κάποια εκδήλωση. Αυτό συμβαίνει για να έχουμε μια μοναδική εγγραφή για έναν συγκεκριμένο άνθρωπο. Δηλαδή για παράδειγμα κάποιος ο οποίος έχει διοργανώσει μια εκδήλωση, να μπορεί να κάνει κράτηση σε κάποια άλλη εκδήλωση μέσω της εφαρμογής για κινητά, με τον ίδιο λογαριασμό. Τα πεδία που έχει το κάθε έγγραφο της συγκεκριμένης συλλογής είναι τα παρακάτω:
    - **_id**: Μοναδικό κλειδί (αναφέρεται ως ObjectId) και δημιουργείται αυτόματα από τη Mongo
    - **email**: Το email του συγκεκριμένου χρήστη
    - **username**: Το όνομα χρήστη
    - **password**: Ο κωδικός του χρήστη
    - **fname**: Το όνομα του χρήστη
    - **lname**: Το επώνυμο του χρήστη
    - **organizerId**: Το συγκεκριμένο πεδίο υποδηλώνει αν ο συγκεκριμένος χρήστης έχει εγγραφεί ως ή και ως διοργανωτής εκδήλωσης και άρα έχει πρόσβαση στην web εφαρμογή των διοργανωτών.



Εικόνα 21: Στιγμιότυπο του collection των users

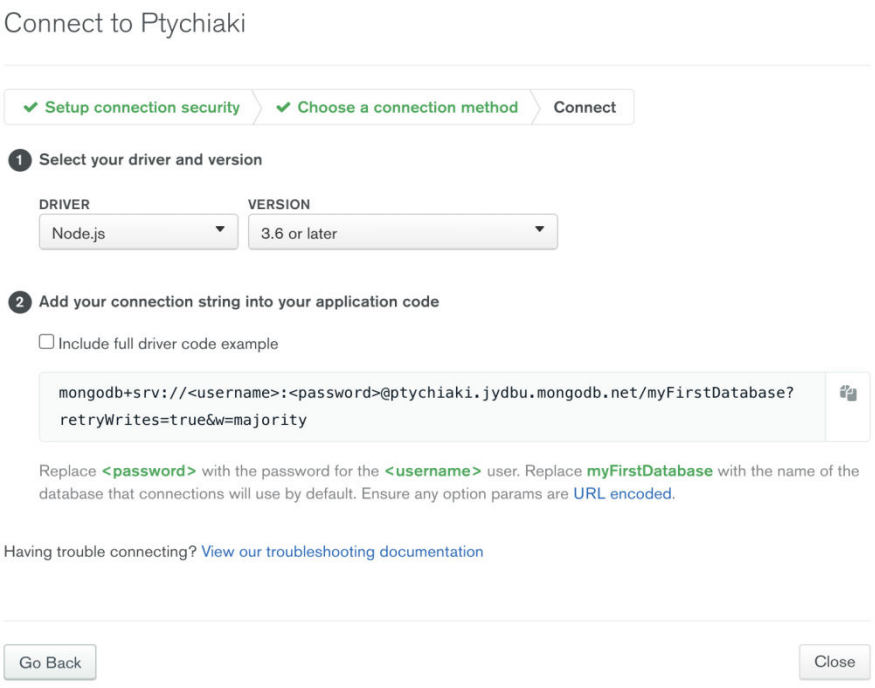
- **enrollments**: Σε αυτή τη συλλογή αποθηκεύονται οι κρατήσεις που πραγματοποιούνται από τους χρήστης της εφαρμογής για κινητά και είναι διαθέσιμες προς προβολή, στα προφίλ των χρηστών των κινητών συσκευών, αλλά και στη web εφαρμογή των διοργανωτών για τη διαχείριση των εκδηλώσεών τους. Τα πεδία που έχει το κάθε έγγραφο της συγκεκριμένης συλλογής είναι τα παρακάτω:
    - **_id:** Μοναδικό κλειδί (αναφέρεται ως ObjectId) και δημιουργείται αυτόματα από τη Mongo
    - **userId:** Το id του χρήστη ο οποίος πραγματοποιεί την κράτηση. Αναφέρεται στη συλλογή των χρηστών
    - **fullName:** Το πλήρες όνομα στο οποίο γίνεται η κράτηση
    - **tickets:** Ο αριθμός των εισιτηρίων
    - **totalPrice:** Η τελική αξία των εισιτηρίων



Εικόνα 22: Στιγμιότυπο του collection των enrollments

4.4 Σύνδεση με τρίτες εφαρμογές

Από τη στιγμή που η βάση δεδομένων έχει δημιουργηθεί και οι συλλογές έχουν οριστικοποιηθεί, πρέπει να οριστεί ο τρόπος με τον οποίο θα επικοινωνεί τόσο με την εφαρμογή για τις κινητές συσκευές, όσο και με την web εφαρμογή για τους διοργανωτές. Και τα δύο αυτά μέρη θα στέλνουν και λαμβάνουν κλήσεις μέσω του API. Το backend του API που έχω υλοποιήσει είναι σε node.js. Η MongoDB έχει τη δυνατότητα να σου δώσει κατευθείαν την εντολή που χρειάζεσαι για να συνδεθεί ένα τρίτο λογισμικό στη βάση δεδομένων. Απλώς επιλέγουμε την έκδοση της node.js που θέλουμε.



Εικόνα 23: Οδηγίες σύνδεσης της MongoDB με το API

4.5 Επίλογος

Στην παρούσα ενότητα αναλύσαμε την δημιουργία, τη δομή και τη διασύνδεση με τρίτες εφαρμογές μιας βάσης δεδομένων σε MongoDB. Το βασικό πλεονέκτημα όπως είδαμε παραπάνω είναι η web πλατφόρμα όπου μπορεί κάποιος database administrator να διαχειρίζεται τη βάση δεδομένων του αλλά και να παίρνει κάποια στατιστικά. Επιπλέον με το connection string που παρέχεται η διασύνδεση με μια εφαρμογή γίνεται πολύ εύκολη. Παρακάτω θα δούμε την ανάπτυξη του REST API αλλά και τη διασύνδεση του με τη βάση δεδομένων.

Κεφάλαιο 5ο: Ανάπτυξη REST Api

5.1 Εισαγωγή

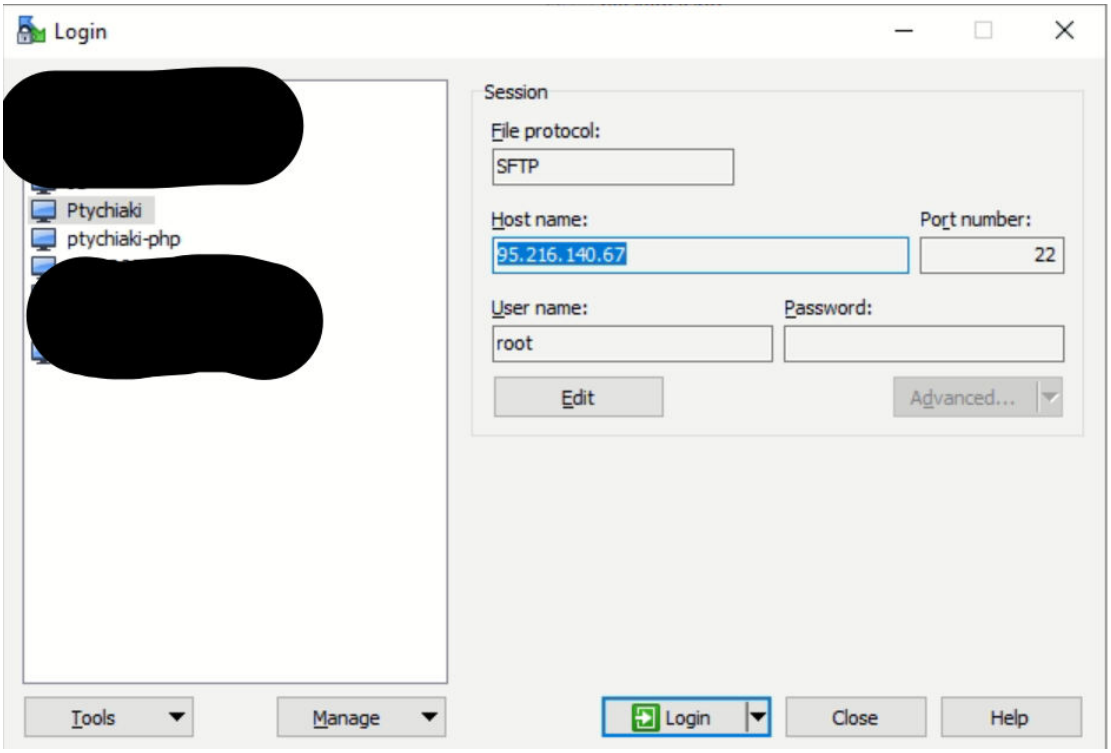
Έχουμε φτάσει στο σημείο όπου έχουμε εγκαταστήσει ότι χρειάζεται ο server που θα στεγάσει το REST Api, όπως επίσης έχουμε δημιουργήσει και τη βάση δεδομένων και έχουμε πάρει τις εντολές τις οποίες πρέπει να γράψουμε στο αρχείο του API, το οποίο είναι υπεύθυνο για τη διασύνδεση με τη βάση δεδομένων.

Τι εννοούμε όμως με τον όρο REST(ful) API; Το κατά κόσμον REST API είναι μια προγραμματιστική διεπαφή όπου μπορεί μια εφαρμογή να στείλει δεδομένα σε μια άλλη με ασφαλή τρόπο χωρίς να χρειάζεται να εγκατασταθεί κάτι. Για παράδειγμα αν μια εκδήλωση δημιουργηθεί στην διαχειριστική web εφαρμογή, μπορεί κάποια τρίτη ιστοσελίδα κράτηση εκδηλώσεων να προβάλλει τη συγκεκριμένη εκδήλωση και σε περίπτωση που γίνει κάποια κράτηση, εκείνη τη στιγμή θα γίνει ένα POST request στο endpoint του API. Αυτό θα έχει ως αποτέλεσμα η πληρότητα της εκδήλωσης να ενημερώνεται στη δική μας βάση δεδομένων και να μη χρειάζεται να υπάρξει συγχρονισμός μεταξύ των δύο βάσεων δεδομένων (της δικής μας και του τρίτου) και άρα να μην υπάρξει υπερπληρότητα σε κάποια εκδήλωση.

Παρακάτω θα δούμε τη διαδικασία που ακολούθησα για τη δημιουργία του API. Για τη σύνδεση με τον server χρησιμοποιώ το λογισμικό WinScp και για τη συγγραφή του κώδικα το λογισμικό Visual Studio Code.

5.2 Σύνδεση στον server

Μέσω του WinScp πραγματοποιώ σύνδεση στον server.



Εικόνα 24: Στιγμιότυπο οθόνης της σύνδεσης σε έναν server μέσω του λογισμικού WinScp

5.3 Δομή των αρχείων

Με το που συνδεθούμε επιτυχώς στον server, το πρώτο βήμα που χρειάζεται να κάνουμε είναι να εγκαταστήσουμε το express framework ώστε να μπορέσουμε με επιτυχία να μετατρέψουμε το VPS που έχουμε νοικιάσει σε έναν λειτουργικό server. Αυτό το επιτυγχάνουμε με τις παρακάτω εντολές:

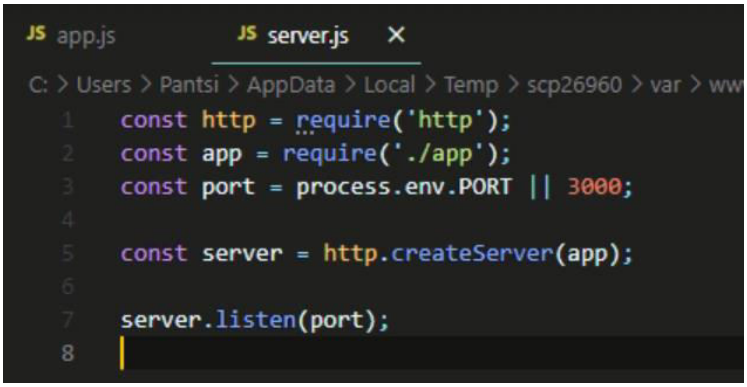
mkdir node-rest-thesis : Η συγκεκριμένη εντολή δημιουργεί έναν φάκελο με όνομα node-rest-thesis μέσα στον οποίο θα δημιουργήσουμε το API

npm init: Η συγκεκριμένη εντολή δημιουργεί το αρχείο package.json και μας ζητάει να επιλέξουμε ποιο θα είναι το αρχείο το οποίο θα εκκινεί την εφαρμογή στέλνοντας μας το εξής response: entry point. Γράφουμε app.js και αυτόματα μας δημιουργεί το αρχείο app.js στο οποίο θα γράψουμε τον σκελετό του API.

npm install express --save: Η συγκεκριμένη εντολή είναι και η τελευταία που πρέπει να εκτελέσουμε και με αυτήν εγκαθίσταται στον server το express framework της javascript.

Επιπλέον πρέπει να ορίσουμε τη δομή που θα έχουν οι φάκελοι και τα αρχεία. Από την εγκατάσταση του Express έχουν δημιουργηθεί τα εξής αρχεία:

- node_modules: Φάκελος ο οποίος περιέχει όλα τα έξτρα πακέτα που έχουμε εγκαταστήσει η θα εγκαταστήσουμε
- server.js: Στο συγκεκριμένο αρχείο αρχικοποιείται ο server και ορίζουμε σε ποια πόρτα θα ακούει.



Εικόνα 25: Στιγμιότυπο του αρχείου server.js

- package.json: Περιλαμβάνει διάφορες πληροφορίες για την εφαρμογή όπως το ποιος είναι ο εκδότης, διάφορα scripts και τα dependencies, δηλαδή όλα τα εξωτερικά πακέτα που έχουν εγκατασταθεί.
- app.js: Το αρχείο της εφαρμογής όπου σε αυτό ορίζουμε ποια endpoints (ποια routes δηλαδή, πχ /users) υπάρχουν και τι θα γίνεται στο καθένα. Παρακάτω θα τα δούμε αναλυτικά.

5.4 App.js

Σε αυτή την ενότητα θα αναλύσουμε σε βάθος το αρχείο app.js με σκοπό να γίνει πλήρως κατανοητός ο τρόπος λειτουργίας του.

```
const express = require('express');
const app = express();
const mongoose = require('mongoose');
//routes
const eventRoutes = require('./api/routes/events');
const userRoutes = require('./api/routes/users');
const enrollmentRoutes = require('./api/routes/enrollments');
//connect to mongo
mongoose.connect('mongodb+srv://root:%23sk33msk%23@ptychiaki.jydbu.mongodb.net/EventApp?retryWrites=true&w=majority',{
    useNewUrlParser: true,
    useUnifiedTopology: true
});
```

Εικόνα 26: Στιγμιότυπο του αρχείο app.js

Στην πρώτη γραμμή ορίζουμε ότι στην εφαρμογή συμπεριλαμβάνεται το express framework και στη δεύτερη γραμμή αρχικοποιούμε την εφαρμογή ως μια εφαρμογή express. Με την τρίτη εντολή, ορίζουμε ότι στην εφαρμογή συμπεριλαμβάνεται το mongoose. Το mongoose είναι μια βιβλιοθήκη της MongoDB και είναι υπεύθυνη για την ομαλή επικοινωνία μιας node.js εφαρμογής και μιας βάσης δεδομένων MongoDB. Να σημειωθεί πως τόσο το express όσο και η mongoose είναι έξτρα πακέτα και έχουν προστεθεί στην εφαρμογή με το npm (node package manager).

Στις επόμενες τρεις γραμμές ορίζουμε τα routes που θα χρησιμοποιήσουμε καθώς και το path στο οποίο βρίσκονται ώστε να συμπεριληφθούν στο app.js. Σε αυτό το σημείο να αναφέρω πως έχει δημιουργηθεί ένας φάκελος με όνομα api, ο οποίος περιέχει τα routes και τα models, τα οποία θα αναλύσουμε στη συνέχεια.

Η επόμενη εντολή είναι αυτή η οποία είχαμε πάρει νωρίτερα από το web περιβάλλον της MongoDB και σε συνδυασμό με τη mongoose καταφέρνουμε να συνδεθούμε επιτυχώς στη βάση δεδομένων.

```
app.use('/events',eventRoutes);
app.use('/users',userRoutes);
app.use('/enrollments',enrollmentRoutes);


app.use((req,res,next) =>{
    const error = new Error('Not Found!');
    error.status = 404;
    next(error);
})

app.use((error,req,res,next)=>{
    res.status(error.status || 500);
    res.json({
        error: {
            message: error.message
        }
    })
})

module.exports = app;
```

Εικόνα 27: Στιγμιότυπο του αρχείου app.js (συνέχεια)

Συνεχίζοντας, οι τρεις πρώτες εντολές του δεύτερου στιγμιότυπου ορίζουν πως όταν κάποιος χρήστης κάνει μια κλήση στο /events για παράδειγμα τότε ο server θα προσπαθήσει να εξυπηρετήσει την κλήση που έγινε, κοιτώντας στο αρχείο ./api/routes/events.

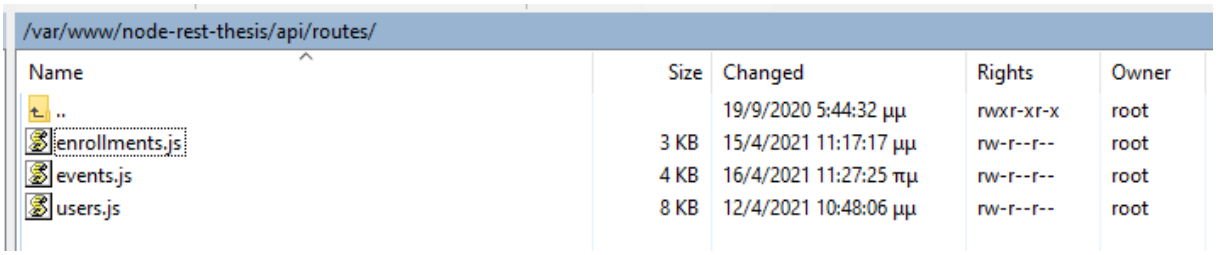
Αν ο εκάστοτε χρήστης κάνει μια κλήση που δεν συμπεριλαμβάνεται σε κάποια από τα τρια παραπάνω routes, τότε θα τρέξουν οι επόμενες εντολές και θα εμφανιστεί το “404 Not Found”. Σε περίπτωση που υπάρχει κάποιο πρόβλημα στον server και δεν μπορέσει να διεκπεραιώσει την κλήση που θα δεχτεί τότε θα τρέξει το τελευταίο μπλοκ εντολών και θα ενημερώσει ότι υπάρχει σφάλμα.

Εφόσον ολοκληρώσαμε τις εντολές και τη δομή του συγκεκριμένου αρχείου, πρέπει να πάμε ένα βήμα παρακάτω και να δούμε τον τρόπο με τον οποίο θα δημιουργήσουμε τα routes και τα models.

5.5 Routes

Στη συγκεκριμένη ενότητα θα αναλύσουμε τα routes τα οποία υπάρχουν στο API. Ο λόγος ύπαρξης των routes, είναι να διαχειρίζονται τα αιτήματα από τις κλήσεις που λαμβάνουν. Μέσω του app.js που αναλύσαμε παραπάνω επιλέγεται σε ποιο route θα πάει κάποια κλήση. Στη συνέχεια μέσα στο route και αναλόγως το όνομα και τον τύπο της κλήσης θα επιλεχθεί και θα εκτελεστεί η κατάλληλη ρουτίνα. Τα routes τα οποία υπάρχουν στην εφαρμογή είναι τα εξής:

- events
- users
- enrollments



Εικόνα 28: Στιγμιότυπο των routes μέσα από την εφαρμογή WinScp

5.5.1 Events Route

Το route των events όπως προδίδει και το όνομα του είναι υπεύθυνο για όλες τις κλήσεις που έχουν να κάνουν με τις εκδηλώσεις. Παρακάτω θα δούμε αναλυτικά τον κώδικα και τις κλήσεις που υπάρχουν διαθέσιμες.

```
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');

const Event = require('../models/event');
```

Εικόνα 29: Στιγμιότυπο των πρώτων γραμμών του αρχείο routes/events.js

Κάθε αρχείο που αφορά route (events,users,enrollments) περιλαμβάνει τις τρεις πρώτες γραμμές. Η πρώτη και η τρίτη γραμμή υποδηλώνουν ότι είναι απαραίτητη η χρήση του express και της mongoose, ενώ η δεύτερη αρχικοποιεί το router με μια έτοιμη εντολή από την express.

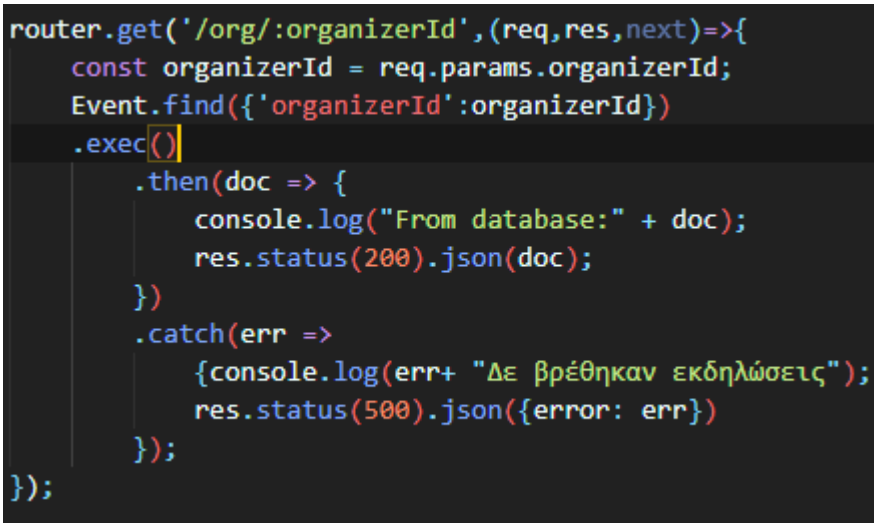
Σχετικά με την τέταρτη εντολή, για να μπορέσουν με επιτυχία να διεκπεραιώνονται οι κλήσεις που θα αφορούν τα events, θα πρέπει να είναι γνωστό το μοντέλο όπως λέγεται. Το μοντέλο δεν είναι τίποτα άλλο από τα πεδία που υπάρχουν διαθέσιμα στις συλλογές της βάσης δεδομένων (collections). Στη συνέχεια θα αναλύσουμε πλήρως το κάθε μοντέλο. Προς το παρόν θα δούμε αναλυτικά το κάθε διαθέσιμο endpoint.

```
router.get('/', (req,res,next)=>{
    Event.find()
        .exec()
        .then(docs =>{
            console.log(docs);
            if(docs.length >= 0){
                res.status(200).json(docs);
            }else{
                res.status(404).json({
                    message: 'Δε βρέθηκαν εκδηλώσεις'
                })
            }

        })
        .catch(err => {
            console.log(err);
            res.status(500).json({
                error: err
            })
        });
});
```

Εικόνα 30: Στιγμιότυπο της μεθόδου GET / από το αρχείο routes/events.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα GET Request στο 95.216.140.67/events/ και επιστρέφει όλες τις εκδηλώσεις που υπάρχουν. Αυτό επιτυγχάνεται κάνοντας αναζήτηση στη βάση δεδομένων με τη μέθοδο .find() και στη συνέχεια εκτελώντας την επιστρέφει τις εκδηλώσεις που υπάρχουν διαθέσιμες ή το μήνυμα 'Δε βρέθηκαν εκδηλώσεις' αν το πλήθος των εγγράφων που θα επιστραφεί είναι μηδέν.



```
router.get('/org/:organizerId',(req,res,next)=>{
    const organizerId = req.params.organizerId;
    Event.find({'organizerId':organizerId})
    .exec()
        .then(doc => {
            console.log("From database:" + doc);
            res.status(200).json(doc);
        })
        .catch(err =>
            {console.log(err+ "Δε βρέθηκαν εκδηλώσεις");
            res.status(500).json({error: err})
        });
});
```

Εικόνα 31: Στιγμιότυπο της μεθόδου GET /org/:organizerId από το αρχείο routes/events.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα GET Request στο 95.216.140.67/events/org/….το id ενός organizer…. και επιστρέφει όλες τις εκδηλώσεις του συγκεκριμένου διοργανωτή. Λειτουργεί με τον ίδιο τρόπο με τον οποίο λειτουργεί και το παραπάνω τμήμα κώδικα, με τη διαφορά ότι εδώ υπάρχει και μια παράμετρος, το organizerId. Στην πρώτη γραμμή του συγκεκριμένου τμήματος κώδικα το βλέπουμε με μια άνω κάτω τελεία στην αρχή του.

Έτσι ορίζονται οι μεταβλητές. Στη συνέχεια η εντολή Event.find() τη λαμβάνει σαν παράμετρο ώστε να επιστρέψει από τη βάση δεδομένων τις εκδηλώσεις του συγκεκριμένου διοργανωτή.

```
router.post('/', (req,res,next)=>{
    const event = new Event({
        _id: new mongoose.Types.ObjectId(),
        title: req.body.title,
        descr: req.body.descr,
        date: req.body.date,
        organizerId: req.body.organizerId,
        venue: req.body.venue,
        capacity: req.body.number,
        price: req.body.price,
        img: req.body.img
    });
    event
        .save()
        .then(result =>{
            console.log(result);
            res.status(201).json({
                message: 'Αποθηκεύτηκε με επιτυχία',
                createdEvent: event
            });
    })
    .catch(err => {
        console.log(err)
        res.status(500).json({
            error: err
        })
    });
});
```

Εικόνα 32: Στιγμιότυπο της μεθόδου POST / από το αρχείο routes/events.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα POST Request στο 95.216.140.67/events/ και εφόσον ο χρήστης περάσει τις σωστές παραμέτρους στην κλήση τότε δημιουργείται και αποθηκεύεται με επιτυχία μια νέα εκδήλωση. Στην αρχή της συγκεκριμένης κλήσης ορίζουμε για το κάθε πεδίο της βάσης δεδομένων σε ποιο πεδίο από το σώμα της κλήσης αντιστοιχεί. Αυτό γίνεται με την εντολή req.body.όνομα_πεδίου. Στο μόνο πεδίο που ορίζουμε κάτι διαφορετικό είναι το _id καθώς παίρνει ένα μοναδικό id, το οποίο δημιουργείται κατά την δημιουργία του εγγράφου. Εφόσον όλα τα πεδία είναι σωστά, τότε εκτελείται η μέθοδος .save() και αποθηκεύεται η νέα εκδήλωση και μας επιστρέφεται το μήνυμα 'Αποθηκεύτηκε με επιτυχία'. Αν κάτι δεν πάει καλά τότε εκτελείται το τελευταίο κομμάτι του κώδικα και μας επιστρέφεται το αντίστοιχο σφάλμα.

```
router.get('/:eventId', (req,res,next)=>{
    const id = req.params.eventId;
    Event.findById(id)
        .exec()
        .then(doc => {
            console.log("From database:" + doc);
            res.status(200).json(doc);
        })
        .catch(err =>
            {console.log(err);
            res.status(500).json({error: err})
        });

});
```

Εικόνα 33: Στιγμιότυπο της μεθόδου GET /:eventId από το αρχείο routes/events.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα GET Request στο 95.216.140.67/:eventId και επιστρέφει μια συγκεκριμένη εκδήλωση. Αυτό επιτυγχάνεται κάνοντας αναζήτηση στη βάση δεδομένων με τη μέθοδο .findById() και στη συνέχεια εκτελώντας την επιστρέφει τα στοιχεία της συγκεκριμένης εκδήλωσης ή διαφορετικά ένα άδειο JSON Object σε περίπτωση που δεν βρεθεί καμία εκδήλωση.

```
router.patch('/:eventId', (req,res,next)=>{
    const id = req.params.eventId;
    const updateOps = {};
    for (const ops of req.body){
        updateOps[ops.propName] = ops.value;
    }
    Event.update({_id:id},{$set: updateOps})
    .exec()
    .then(result => {
        console.log(result);
        res.status(200).json(result);
    })
    .catch(err =>{
        console.log(err);
        res.status(500).json({
            error:err
        });
    });
});
```

Εικόνα 34: Στιγμιότυπο της μεθόδου PATCH /:eventId από το αρχείο routes/events.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης (διοργανωτής) κάνει ένα PATCH Request στο 95.216.140.67/:eventId. Με την μέθοδο PATCH μπορούμε να μεταβάλλουμε τις πληροφορίες ενός εγγράφου της βάσης δεδομένων, για παράδειγμα όταν ο διοργανωτής θέλει να

αλλάξει τον αριθμό της χωρητικότητας μιας εκδήλωσης. Η μέθοδος της mongoose που χρησιμοποιούμε είναι η .update() και ως παραμέτρους παίρνει το id και τα πεδία τα οποία θέλουμε να μεταβάλλουμε, με τις νέες τιμές.

```
router.delete('/:eventId', (req,res,next)=>{
    const id = req.params.eventId;
    Event.remove({_id: id})
        .exec()
        .then(res =>{
            res.status(200).json(result);
        })
        .catch(err =>{
            console.log(err);
            res.status(500).json({
                error: err
            })
        });
});
```

Εικόνα 35: Στιγμιότυπο της μεθόδου DELETE /:eventId από το αρχείο routes/events.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης (διοργανωτής) κάνει ένα DELETE Request στο 95.216.140.67/:eventId. Με την μέθοδο DELETE μπορούμε να διαγράψουμε ένα έγγραφο εκδήλωσης στη βάση δεδομένων. Η μέθοδος της mongoose που χρησιμοποιούμε είναι η .remove() και ως παράμετρο παίρνει το id του event που θέλουμε να διαγράψουμε.

Τα παραπάνω είναι τα διαθέσιμα endpoints που υπάρχουν σχετικά με τα events.

5.5.2 Users Route

Παρακάτω θα δούμε τα endpoints που αφορούν τους χρήστες (users)

```
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');

const User = require('../models/user');
```

Εικόνα 36: Στιγμιότυπο των πρώτων γραμμών του αρχείο routes/users.js

Όπως και παραπάνω έτσι και εδώ οι τρεις πρώτες εντολές είναι οι ίδιες. Η διαφορά που παρατηρούμε βρίσκεται στην τέταρτη γραμμή. Πρόκειται για την βιβλιοθήκη bcrypt, η οποία είναι υπεύθυνη στο να κάνει hash κωδικούς. Αυτό έχει ως αποτέλεσμα να μην αποθηκεύονται plain text στη βάση δεδομένων και έτσι να τηρούνται κάποιες βασικές αρχές της ασφάλειας πληροφοριακών συστημάτων.

Η τελευταία εντολή όπως και παραπάνω είναι το μοντέλο των χρηστών που έχουμε δημιουργήσει και θα δούμε στη συνέχεια.

```
router.post('/signup', (req,res,next)=>{
    User.find({username:req.body.username})
      .exec()
      .then(user=>{
          if(user.length >= 1){
              return res.status(409).json({
                  message:'Το όνομα χρήστη υπάρχει ήδη'
              });
          }else{
            bcrypt.hash(req.body.password,10,(err,hash)=>{
                if(err){
                    return res.status(500).json({
                        error: err
                    });
                }else{
                    const user = new User({
                        _id: new mongoose.Types.ObjectId(),
                        username: req.body.username,
                        password: hash
                    });
                    user
                      .save()
                      .then(result =>{
                          console.log(result);
                          res.status(201).json({
                              message: 'Ο χρήστης δημιουργήθηκε'
                          });
                      })
                      .catch(err=>{
                          console.log(err);
                          res.status(500).json({
                              error:err
                          });
                      });
                }
            })
          }
      })
});
```

Εικόνα 37: Στιγμιότυπο της μεθόδου POST/signup από το αρχείο routes/users.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης της εφαρμογής θέλει να δημιουργήσει έναν νέο λογαριασμό. Η διαδικασία που γίνεται είναι ότι αναζητείται στη βάση δεδομένων αν υπάρχει το συγκεκριμένο username και στην περίπτωση που δεν υπάρχει τότε γίνεται η δημιουργία του username και του password που έχει θέσει ο χρήστης. Το password όπως

προαναφέραμε δημιουργείται μέσω του bcrypt και γίνεται hash. Εφόσον όλα εκτελεστούν επιτυχώς τότε δημιουργείται ο χρήστης, διαφορετικά λαμβάνουμε το αντίστοιχο σφάλμα.

```
router.post('/login',(req,res,next)=>{
    User.find({username:req.body.username})
      .exec()
      .then(user => {
          if(user.length < 1){
              return res.status(401).json({
                  message:'Auth failed'
              });
          }
          bcrypt.compare(req.body.password,user[0].password,(err,result)=>{
              if(err){
                  return res.status(401).json({
                      message:'Auth failed'
                  });
              }
              if(result){
                const token = jwt.sign({
                    username: user[0].username,
                    userId: user[0]._id
                },
                process.env.JWT_KEY,
                {
                    expiresIn: "1h"
                }
                );
                return res.status(200).json({
                    message:'Auth successful',
                    token: token,
                    userId: user[0]._id
                })
              }
              res.status(401).json({
                message:'Auth failed'
              });
          })
      })
      .catch(err =>{
            console.log(err);
            res.status(500).json({
                error: err
            });
      });
});
```

Εικόνα 38: Στιγμιότυπο της μεθόδου POST/signup από το αρχείο routes/users.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης της εφαρμογής θέλει να συνδεθεί. Στην αρχή γίνεται αναζήτηση του username στη βάση δεδομένων. Σε περίπτωση που υπάρχει, άρα ο χρήστης είναι υπαρκτός τότε με τη χρήση της bcrypt γίνεται σύγκριση του unhashed password πλέον, και σε περίπτωση που ταιριάζει τότε επιτρέπει στον χρήστη να εισέλθει στην εφαρμογή. Διαφορετικά εμφανίζονται τα αντίστοιχα μηνύματα.

```
router.get('/', (req,res,next)=>{
    User.find()
        .exec()
        .then(docs =>{
            console.log(docs);
            if(docs.length >= 0){
                res.status(200).json(docs);
            }else{
                res.status(404).json({
                    message: 'Δε βρέθηκαν χρήστες'
                })
            }

        })
        .catch(err => {
            console.log(err);
            res.status(500).json({
                error: err
            })
        });
});
```

Εικόνα 39: Στιγμιότυπο της μεθόδου GET/ από το αρχείο routes/users.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα GET Request στο 95.216.140.67/users/ και επιστρέφει όλους τους χρήστες που υπάρχουν. Αυτό επιτυγχάνεται κάνοντας αναζήτηση στη βάση δεδομένων με τη μέθοδο .find() και στη συνέχεια εκτελώντας την επιστρέφει τους χρήστες που υπάρχουν ή το μήνυμα 'Δε βρέθηκαν χρήστες' αν το πλήθος των εγγράφων που θα επιστραφεί είναι μηδέν.

```
router.get('/:username',(req,res,next)=>{
    const username = req.params.username;
    User.find({'username':username},{email:1,username:1,fname:1,lname:1})
    .exec()
        .then(doc => {
            console.log("From database:" + doc);
            res.status(200).json(doc);
        })
        .catch(err =>
            {console.log(err);
            res.status(500).json({error: err})
        });
});
```

Εικόνα 40: Στιγμιότυπο της μεθόδου GET/:username από το αρχείο routes/users.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα GET Request στο 95.216.140.67/:usename και επιστρέφει έναν χρήστη με το συγκεκριμένο username, αν αυτός υπάρχει. Αυτό επιτυγχάνεται κάνοντας αναζήτηση στη βάση δεδομένων με τη μέθοδο .find() και ως παράμετρο λαμβάνει το username και στη συνέχεια με το όνομα του πεδίου και έναν 1 ορίζουμε τα πεδία που θέλουμε να επιστραφούν. Στην περίπτωση μας ζητάμε να μας επιστρέψει το email, το username, το όνομα και το επώνυμο του χρήστη, αν αυτός υπάρχει. Διαφορετικά μας επιστρέφει ένα άδειο JSON Object.

```
router.patch('/:userId', (req,res,next)=>{
    const id = req.params.userId;
    const updateOps = {};
    for (const ops of req.body){
        updateOps[ops.propName] = ops.value;
    }
    User.update({_id:id},{$set: updateOps})
    .exec()
    .then(result => {
        console.log(result);
        res.status(200).json(result);
    })
    .catch(err =>{
        console.log(err);
        res.status(500).json({
            error:err
        });
    });
});
```

Εικόνα 41: Στιγμιότυπο της μεθόδου PATCH/:userId από το αρχείο routes/users.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης (διοργανωτής) κάνει ένα PATCH Request στο 95.216.140.67/:userId. Με την μέθοδο PATCH μπορούμε να μεταβάλλουμε τις πληροφορίες ενός εγγράφου της βάσης δεδομένων, για παράδειγμα όταν ένας χρήστης θέλει να αλλάξει κάποιο από τα στοιχεία του. Η μέθοδος της mongoose που χρησιμοποιούμε είναι η .update() και ως παραμέτρους παίρνει το id και τα πεδία τα οποία θέλουμε να μεταβάλλουμε, με τις νέες τιμές.

```
router.delete('/:userId', (req,res,next)=>{
    const id = req.params.userId;
    User.remove({_id: id})
        .exec()
        .then(res =>{
            res.status(200).json(result);
        })
        .catch(err =>{
            console.log(err);
            res.status(500).json({
                error: err
            })
        });
});
```

Εικόνα 42: Στιγμιότυπο της μεθόδου DELETE/:userId από το αρχείο routes/users.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα DELETE Request στο 95.216.140.67/:eventId. Με την μέθοδο DELETE μπορούμε να διαγράψουμε ένα έγγραφο χρήστη στη

βάση δεδομένων. Η μέθοδος της mongoose που χρησιμοποιούμε είναι η .remove() και ως παράμετρο παίρνει το id του user που θέλουμε να διαγράψουμε.

Τα παραπάνω είναι τα διαθέσιμα endpoints που υπάρχουν σχετικά με τα events.

5.5.3 Enrollments Route

Παρακάτω θα δούμε τα διαθέσιμα endpoints που αφορούν τα enrollments (κρατήσεις)

```
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');

const Enrollment = require('../models/enrollment');
```

Εικόνα 43: Στιγμιότυπο των πρώτων γραμμών του αρχείο routes/enrollments.js

Όπως και παραπάνω έτσι και εδώ οι τρεις πρώτες εντολές είναι οι ίδιες και η τέταρτη αφορά το μοντέλο.

```
router.get('/', (req,res,next)=>{
    Enrollment.find()
        .exec()
        .then(docs =>{
            console.log(docs);
            if(docs.length >= 0){
                res.status(200).json(docs);
            }else{
                res.status(404).json({
                    message: 'Δεν υπάρχει καμία κράτηση
                })
            }

        })
        .catch(err => {
            console.log(err);
            res.status(500).json({x
                error: err
            })
        });
});
```

Εικόνα 44: Στιγμιότυπο της μεθόδου GET/ από το αρχείο routes/enrollments.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα GET Request στο 95.216.140.67/enrollments/ και επιστρέφει όλες τις κρατήσεις που υπάρχουν. Αυτό επιτυγχάνεται κάνοντας αναζήτηση στη βάση δεδομένων με τη μέθοδο .find() και στη συνέχεια εκτελώντας την επιστρέφει τους χρήστες που υπάρχουν ή το μήνυμα 'Δεν υπάρχει καμία κράτηση' αν το πλήθος των εγγράφων που θα επιστραφεί είναι μηδέν.

```
router.get('/:userId',(req,res,next)=>{
    const userId = req.params.userId;
    Enrollment.find({'userId':userId})
    .exec()
        .then(doc => {
            console.log("From database:" + doc);
            res.status(200).json(doc);
        })
        .catch(err =>
            {console.log(err);
            res.status(500).json({error: err})
        });
});
```

Εικόνα 45: Στιγμιότυπο της μεθόδου GET/:userId από το αρχείο routes/enrollments.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα GET Request στο 95.216.140.67/:userId και επιστρέφει τις κρατήσεις ενός συγκεκριμένου χρήστη. Αυτό επιτυγχάνεται κάνοντας αναζήτηση στη βάση δεδομένων με τη μέθοδο .find() και με παράμετρο την userId. Στη συνέχεια εκτελώντας την επιστρέφει τις κρατήσεις του χρήστη ή διαφορετικά ένα άδειο JSON Object σε περίπτωση που δεν βρεθεί καμία κράτηση.

```
router.get('/evt/:eventId',(req,res,next)=>{
    const eventId = req.params.eventId;
    Enrollment.find({'eventId':eventId})
    .exec()
        .then(doc => {
            console.log("From database:" + doc);
            res.status(200).json(doc);
        })
        .catch(err =>
            {console.log(err);
            res.status(500).json({error: err})
        });
});
```

Εικόνα 46: Στιγμιότυπο της μεθόδου GET/evt/:eventId από το αρχείο routes/enrollments.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης κάνει ένα GET Request στο 95.216.140.67/evt/:eventId και επιστρέφει τις κρατήσεις που έχουν γίνει σε μια συγκεκριμένη εκδήλωση. Αυτό επιτυγχάνεται κάνοντας αναζήτηση στη βάση δεδομένων με τη μέθοδο .find() και με παράμετρο την eventId. Στη συνέχεια εκτελώντας την επιστρέφει τις κρατήσεις της συγκεκριμένης εκδήλωσης ή διαφορετικά ένα άδειο JSON Object σε περίπτωση που δεν βρεθεί καμία κράτηση.

```
router.post('/', (req,res,next)=>{
    const enrollment = new Enrollment({
        _id: new mongoose.Types.ObjectId(),
        eventId: req.body.eventId,
        userId: req.body.userId,
        fullname: req.body.fullname,
        tickets: req.body.tickets,
        totalPrice: req.body.totalPrice,
        bookDate: req.body.bookDate
    });
    enrollment
        .save()
        .then(result =>{
            console.log(result);
            res.status(201).json({
                message: 'Αποθηκεύτηκε με επιτυχία',
                createdEnrollment: enrollment
            });
    })
    .catch(err => {
        console.log(err)
        res.status(500).json({
            error: err
        })
    });
});
```

Εικόνα 47: Στιγμιότυπο της μεθόδου POST/ από το αρχείο routes/enrollments.js

Το συγκεκριμένο τμήμα κώδικα εκτελείται όταν κάποιος χρήστης της εφαρμογής για κινητά κάνει κράτηση σε κάποια εκδήλωση.

Εφόσον ο χρήστης περάσει τις σωστές παραμέτρους στην κλήση τότε δημιουργείται και αποθηκεύεται με επιτυχία μια νέα κράτηση. Στην αρχή της συγκεκριμένης κλήσης ορίζουμε για το κάθε πεδίο της βάσης δεδομένων σε ποιο πεδίο από το σώμα της κλήσης αντιστοιχεί. Αυτό γίνεται με την εντολή req.body.όνομα_πεδίου. Στο μόνο πεδίο που ορίζουμε κάτι διαφορετικό είναι το _id καθώς παίρνει ένα μοναδικό id, το οποίο δημιουργείται κατά την δημιουργία του εγγράφου. Εφόσον όλα τα πεδία είναι σωστά, τότε εκτελείται η μέθοδος .save() και αποθηκεύεται η νέα κράτηση και μας επιστρέφεται το μήνυμα ‘Αποθηκεύτηκε με επιτυχία’. Αν κάτι δεν πάει καλά τότε εκτελείται το τελευταίο κομμάτι του κώδικα και μας επιστρέφεται το αντίστοιχο σφάλμα.

Όπως έχουμε δει σε όλα τα παραπάνω endpoints, η αρχιτεκτονική του API είναι τέτοια, που είναι απαραίτητο να δημιουργηθούν τα αντίστοιχα models, τη δημιουργία των οποίων θα δούμε στην αμέσως επόμενη ενότητα.

5.6 Models

Σε αυτή την ενότητα θα δούμε τα models της κάθε οντότητας (events,users,enrollments). Η χρήση των models είναι ώστε να γνωρίζουν τα endpoints ποια πεδία υπάρχουν διαθέσιμα στην βάση και να διαχειρίζονται όπως πρέπει τις κλήσεις

5.6.1 Events

```
const mongoose = require('mongoose');

const eventSchema = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,
    title: String,
    descr: String,
    date: Date,
    organizerId: String,
    venue: String,
    capacity: Number
});


module.exports = mongoose.model('Event',eventSchema);
```

Εικόνα 48: Στιγμιότυπο του event model από το αρχείο models/event.js

Το συγκεκριμένο τμήμα κώδικα ορίζει τη μορφή του event model. Στην πρώτη γραμμή, όπως και στα routes ορίζουμε ότι χρησιμοποιείται η mongoose. Στην τρίτη γραμμή ορίζουμε ότι το eventSchema είναι ένα σχήμα τύπου mongoose με τα εξής πεδία:

- _id: τύπου ObjectId
- title: τύπου String
- descr: τύπου String
- date: τύπου Date
- organizerId: τύπου String
- venue: τύπου String
- capacity: τύπου Number

Τα παραπάνω είναι τα πεδία που έχουμε αναλύσει παραπάνω στο κεφάλαιο της βάσης δεδομένων. Εδώ το μόνο που χρειάζεται είναι να ορίσουμε το όνομα τους και τον τύπο τους.

Στην τελευταία εντολή ορίζουμε ότι το Event είναι ένα model τύπου eventSchema και μπορεί να χρησιμοποιηθεί έτσι στα routes που αναλύσαμε παραπάνω.

5.6.2 Users

```
const mongoose = require('mongoose');

const userSchema = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,
    username: {type:String,required:true, unique: true},
    password: {type:String,required:true},
    fname: String,
    lname: String
});

module.exports = mongoose.model('User',userSchema);
```

Εικόνα 49: Στιγμιότυπο του user model από το αρχείο models/user.js

Το συγκεκριμένο τμήμα κώδικα ορίζει τη μορφή του user model. Στην πρώτη γραμμή, όπως και στα routes ορίζουμε ότι χρησιμοποιείται η mongoose. Στην τρίτη γραμμή ορίζουμε ότι το userSchema είναι ένα σχήμα τύπου mongoose με τα εξής πεδία:

- _id: τύπου ObjectId
- username: τύπου String, υποχρεωτικό και μοναδικό
- password: τύπου String, υποχρεωτικό
- fname: τύπου String
- lname: τύπου String

Τα παραπάνω είναι τα πεδία που έχουμε αναλύσει παραπάνω στο κεφάλαιο της βάσης δεδομένων. Εδώ το μόνο που χρειάζεται είναι να ορίσουμε το όνομα τους και τον τύπο τους.

Στην τελευταία εντολή ορίζουμε ότι ο User είναι ένα model τύπου userSchema και μπορεί να χρησιμοποιηθεί έτσι στα routes που αναλύσαμε παραπάνω.

5.6.3 Enrollments

```
const mongoose = require('mongoose');

const enrollmentSchema = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,
    eventId: String,
    userId: String,
    fullname: String,
    tickets: Number,
    totalPrice: Number,
    bookDescr: Date
});

module.exports = mongoose.model('Enrollment',enrollmentSchema);
```

Εικόνα 50: Στιγμιότυπο του enrollment model από το αρχείο models/enrollment.js.

Το συγκεκριμένο τμήμα κώδικα ορίζει τη μορφή του user model. Στην πρώτη γραμμή, όπως και στα routes ορίζουμε ότι χρησιμοποιείται η mongoose. Στην τρίτη γραμμή ορίζουμε ότι το userSchema είναι ένα σχήμα τύπου mongoose με τα εξής πεδία:

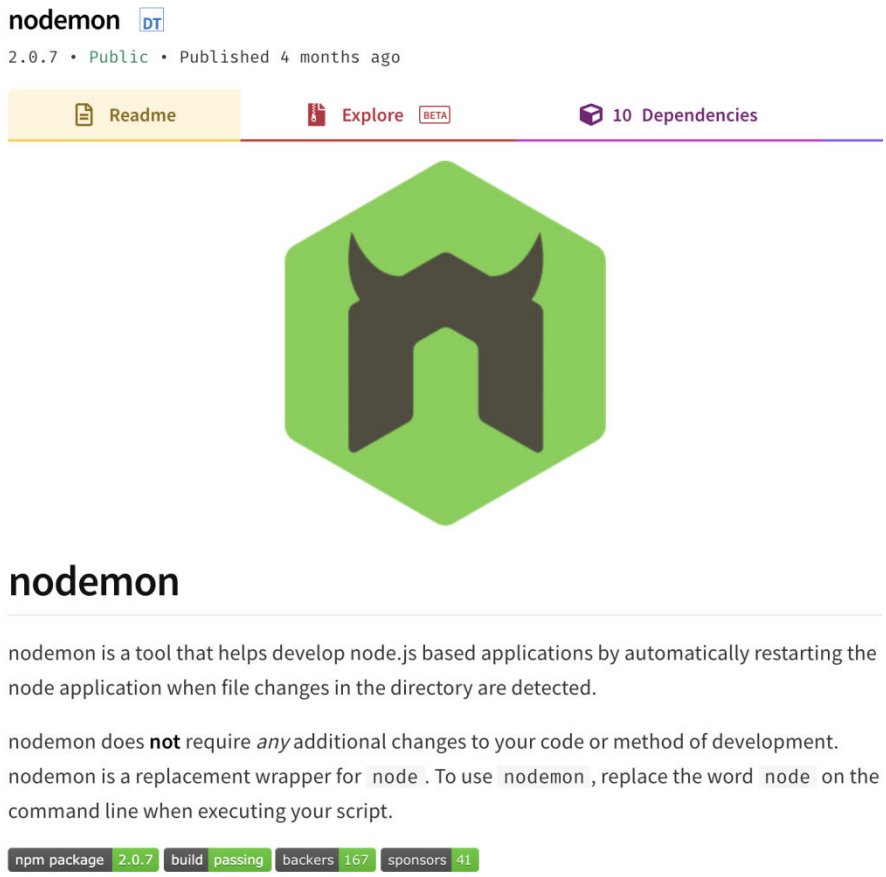
- _id: τύπου ObjectId
- eventId: τύπου String
- userId: τύπου String
- fullname: τύπου String
- tickets: τύπου Number
- totalPrice: τύπου Number
- bookDescr: τύπου Date

Τα παραπάνω είναι τα πεδία που έχουμε αναλύσει παραπάνω στο κεφάλαιο της βάσης δεδομένων. Εδώ το μόνο που χρειάζεται είναι να ορίσουμε το όνομα τους και τον τύπο τους.

Στην τελευταία εντολή ορίζουμε ότι το Enrollment είναι ένα model τύπου enrollmentSchema και μπορεί να χρησιμοποιηθεί έτσι στα routes που αναλύσαμε παραπάνω.

5.7 Nodemon

Από όλα τα dependencies που χρησιμοποίησα στον συγκεκριμένο server, το σημαντικότερο και αυτό που αξίζει να αναφερθεί είναι το nodemon.



Εικόνα 51: Στιγμιότυπο του nodemon από το npmjs.com

Επί της ουσίας το nodemon είναι υπεύθυνο να ελέγχει κάθε στιγμή τον server και μόλις αντιληφθεί οποιαδήποτε αλλαγή συνέβη, είτε από τον προγραμματιστή είτε από κάποιον άλλο πόρο, να κάνει επανεκκίνηση στον server.

Με το συγκεκριμένο dependency κατάφερα να γλιτώσω αρκετές εργατοώρες καθώς πέρα από την αυτόματη επανεκκίνηση, όταν κάτι δεν είχε πάει καλά στον κώδικα μου, με ενημέρωνε ακριβώς για το που βρίσκεται το σφάλμα και έτσι μπορούσα να κάνω πιο γρήγορο debugging.

Από τη στιγμή που έχει ολοκληρωθεί το API και ο server έχει τις σωστές ρυθμίσεις, είναι η στιγμή να περάσουμε στην ανάπτυξη της web εφαρμογής διαχείρισης εκδηλώσεων.

5.8 Επίλογος

Ο τρόπος με τον οποίο αναπτύχθηκε το API είναι τέτοιος που στο μέλλον μπορεί πολύ εύκολα να προστεθεί κάποιο καινούργιο route με μερικές μόνο γραμμές κώδικα. Αυτό έχει ως αποτέλεσμα να είναι πιο βατή η διασύνδεση του σε άλλες εφαρμογές. Από τη στιγμή που έχει ολοκληρωθεί το API και ο server έχει τις σωστές ρυθμίσεις, είναι η στιγμή να περάσουμε στην ανάπτυξη της web εφαρμογής διαχείρισης εκδηλώσεων.

Κεφάλαιο 6ο: Ανάπτυξη Web εφαρμογής διαχείρισης εκδηλώσεων

6.1 Εισαγωγή

Στη συγκεκριμένη ενότητα θα αναλύσουμε τον τρόπο υλοποίησης της web εφαρμογής διαχείρισης εκδηλώσεων. Όπως έχει προαναφερθεί ο server στον οποίο θα αναπτυχθεί, χρησιμοποιεί ubuntu 20.04, και έχει εγκατασταθεί PHP και Apache2. Ο λόγος για τον οποίο έχω επιλέξει την PHP για τη συγκεκριμένη εφαρμογή είναι για να εμβαθύνω στη λειτουργία του session της PHP καθώς η node.js δεν έχει εγγενώς το session.

Όπως έχω αναφέρει παραπάνω για τη συγκεκριμένη εφαρμογή έχει χρησιμοποιηθεί ένα έτοιμο bootstrap template (sbadmin 2). Ο λόγος για τον οποίο έχει χρησιμοποιηθεί ένα τέτοιου είδους template είναι ώστε εύκολα και γρήγορα να μπορούν να προστεθούν νέες δυνατότητες στο ui ενός διαχειριστή εκδηλώσεων. Η ομάδα ανάπτυξης του συγκεκριμένου template έχει δημιουργήσει με css διάφορα custom πράγματα, όπως για παράδειγμα buttons, τα οποία μπορούν να χρησιμοποιηθούν με συγκεκριμένο τρόπο, όπως αναφέρουν στο documentation που υπάρχει διαθέσιμο. Ο συνηθέστερος τρόπος που είναι υλοποιημένα τέτοιου είδους templates, είναι ότι ορίζουν κάποιες css κλάσεις, τις οποίες μπορούμε να χρησιμοποιήσουμε και αυτόματα τα html elements να αποκτήσουν το styling που έχει οριστεί.

Τα δύο βασικά αρχεία που χρησιμοποιούνται από το template, είναι το login.html και το index.html. Επιπλέον έχω δημιουργήσει ένα αρχείο php, το start.php το οποίο είναι το κύριο αρχείο της εφαρμογής.

6.2 start.php

Το αρχείο start.php είναι αυτό το οποίο είναι υπεύθυνο για τον έλεγχο του session και κατ επέκταση του χρήστη ο οποίος είναι συνδεδεμένος σε κάποιον συγκεκριμένο υπολογιστή κάποια συγκεκριμένη χρονική στιγμή. Αυτό που διαφοροποιεί τους χρήστες είναι το usrid, δηλαδή το μοναδικό id που έχει κάποιος χρήστης.

start.php

```php
<?php
session_start();
error_reporting(E_ALL | E_STRICT);
ini_set('display_errors', 'On');

function validateLogin($myPage) {
    $curl = curl_init();
    $sp="{\n\t\"username\":\"myusername\",\n\t\"password\":\"mypass\"\n}";
    $sp = str_replace("myusername",$_POST["memail"],$sp);
    $sp = str_replace("mypass",$_POST["mpass"],$sp);
    curl_setopt_array($curl, array(
      CURLOPT_URL => "http://95.216.140.67:3000/users/login",
      CURLOPT_RETURNTRANSFER => true,
      CURLOPT_ENCODING => "",
      CURLOPT_MAXREDIRS => 10,
      CURLOPT_TIMEOUT => 0,
      CURLOPT_FOLLOWLOCATION => true,
      CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
      CURLOPT_CUSTOMREQUEST => "POST",
      CURLOPT_POSTFIELDS =>$sp,
      CURLOPT_HTTPHEADER => array("Content-Type: application/json"),
    ));
    $response = curl_exec($curl);
    curl_close($curl);
    $r = json_decode($response, true);
    echo $r["message"];
    if ($r["message"] == "Auth successful") $_SESSION["usrid"] = $r["userId"]; else session_unset();
}

```

Εικόνα 52: Στιγμιότυπο του αρχείου start.php της εφαρμογής διαχείρισης εκδηλώσεων.

Στην πρώτη γραμμή υπάρχει η ετικέτα έναρξης της php, που υποδηλώνει ότι από εκεί και πέρα ακολουθεί php κώδικας. Η δεύτερη γραμμή ορίζει ότι ξεκινάει το session και η χρήση των γραμμών τρία και τέσσερα είναι ώστε να εμφανίζονται τυχόν σφάλματα στον browser, ώστε να επιτυγχάνεται ευκολότερα το debugging.

Στη συνέχεια ακολουθεί η συνάρτηση validateLogin η οποία είναι διαθέσιμη προς κλήση από οποιοδήποτε άλλο αρχείο. Επί της ουσίας το αρχείο index.html θα την καλεί ώστε να μπορεί κάποιος χρήστης να κάνει σύνδεση στην εφαρμογή. Ο τρόπος με τον οποίο έχει υλοποιηθεί είναι με την βιβλιοθήκη curl, η οποία επιτρέπει σε κάποιον να κάνει HTTP κλήσεις μέσω της PHP. Καταρχάς πρέπει να αρχικοποιήσουμε την curl και στη συνέχεια σε μια μεταβλητή (στην προκειμένη περίπτωση την sp) να ορίσουμε αυτά τα οποία θα συμπεριλάβουμε στο POST request (στην προκειμένη περίπτωση το username και το password.) Στη συνέχεια αντικαθιστούμε το username και το password με αυτά τα οποία έχει εισάγει ο χρήστης. Έπειτα υπάρχει ο πίνακας των ορισμάτων ή αλλιώς curl_setopt_array. Αξίζει να αναφέρουμε ένα προς ένα τα ορίσματα που υπάρχουν σε ένα curl request.

- CURLOPT_URL: Η συγκεκριμένη παράμετρος ορίζει το URL στο οποίο γίνεται η κλήση. Το συγκεκριμένο είναι url το οποίο έχουμε δημιουργήσει στο API service.
- CURLOPT_RETURNTRANSFER: Η συγκεκριμένη παράμετρος ορίζεται true αν θέλουμε μετά την εκτέλεση να μας επιστραφεί και το αποτέλεσμα σε μορφή συμβολοσειράς (string)
- CURLOPT_ENCODING: Υποχρεωτική παράμετρος όπου μπορούμε να ορίσουμε την κωδικοποίηση της απάντησης, ώστε όταν επιστραφεί στο τέλος να γνωρίζει η CURL αν πρέπει να κάνει αποκωδικοποίηση ή οχι.
- CURLOPT_MAXREDIRS: Η συγκεκριμένη παράμετρος ορίζει το μέχρι πόσες ανακατευθύνσεις μπορούν να συμβούν.
- CURLOPT_TIMEOUT: Η συγκεκριμένη παράμετρος ορίζει το μέγιστο χρονικό διάστημα που επιτρέπεται να διαρκέσει η κλήση. Αν παρέλθει αυτό το χρονικό διάστημα τότε η συγκεκριμένη κλήση ακυρώνεται.
- CURLOPT_FOLLOWLOCATION: Η συγκεκριμένη παράμετρος στέλνεται στο κομμάτι της κεφαλίδας της HTTP κλήσης. Ορίζει αν η κλήση θα "ακολουθήσει" οποιοδήποτε μονοπάτι ή συγκεκριμένο.
- CURLOPT_HTTP_VERSION: Η συγκεκριμένη παράμετρος ορίζει την έκδοση του HTTP πρωτοκόλλου που θα χρησιμοποιηθεί στην κλήση.
- CURLOPT_CUSTOMREQUEST: Η συγκεκριμένη παράμετρος ορίζει τον τύπο της κλήσης, για παράδειγμα (GET,POST)
- CURLOPT_POSTFIELDS: Η συγκεκριμένη παράμετρος συγκεντρώνει τα ορίσματα τα οποία θα αποσταλούν.
- CURLOPT_HTTPHEADER: Η συγκεκριμένη παράμετρος ορίζει τον τύπο του περιεχομένου της κλήσης. Στην περίπτωση μας το περιεχόμενο είναι της μορφής JSON (Javascript Object Notation)

Για να πραγματοποιήσουμε μια CURL κλήση δεν χρειάζονται κάποιες επιπλέον παράμετροι. Βέβαια υπάρχουν δεκάδες ακόμη, οι οποίες όμως είναι προαιρετικές.

Έπειτα η απάντηση της κλήσης αποθηκεύεται σε μια μεταβλητή (στην περίπτωση μας αυτή η μεταβλητή ονομάζεται response). Στην επόμενη γραμμή τερματίζουμε την curl κλήση και στη συνέχεια σε μια μεταβλητή πχ r αποκωδικοποιούμε την απάντηση σε json μορφή. Έτσι μπορούμε να ελέγξουμε αν υπάρχει ο συγκεκριμένος χρήστης και να γίνει σύνδεση με επιτυχία. Εφόσον έχουμε αναλύσει την curl κλήση, τώρα θα δούμε τον τρόπο με τον οποίο καλείται. Αυτό συμβαίνει στο login.html

6.3 login.html

Το login.html είναι η σελίδα στην οποία ανακατευθύνεται κάποιος ο οποίος δεν έχει συνδεθεί στην εφαρμογή. Ο λόγος ύπαρξης της είναι περισσότερο από πασιφανής. Υπάρχει για να επιτρέπει στους χρήστες να συνδέονται.

Από το συγκεκριμένο αρχείο, άξια αναφοράς είναι δύο σημεία. Η υλοποίηση της φόρμας σύνδεσης και η συνάρτηση η οποία καλείται ώστε να επιτευχθεί η ajax κλήση προς το start.php.

```
<div class="text-center">
    <h1 class="h4 text-gray-900 mb-4">Σύνδεση</h1>
</div>
<form class="user" method=post action=start.php >
    <div class="form-group">
        <input type="text" class="form-control form-control-user"
            id="emailInput" name="memail" aria-describedby="emailHelp"
            placeholder="Email">
    </div>
    <div class="form-group">
        <input type="password" class="form-control form-control-user"
            id="passwordInput" name="mpass" placeholder="Κωδικός">
    </div>
    <div class="form-group">
        <div class="custom-control custom-checkbox small">
            <input type="checkbox" class="custom-control-input" id="customCheck">
            <label class="custom-control-label" for="customCheck">Να με θυμάσαι</label>
        </div>
    </div>
    <!-- <a onclick="ulogin()" class="btn btn-primary btn-user btn-block">
        Σύνδεση
    </a> -->
    <input class="btn btn-primary btn-user btn-block" type=submit value="Σύνδεση">
</form>
```

Εικόνα 53: Στιγμιότυπο του αρχείου login.html της εφαρμογής διαχείρισης εκδηλώσεων.

Ο τρόπος με τον οποίο έχει υλοποιηθεί το login είναι μια html φόρμα η οποία σαν μέθοδο έχει την post και σαν αρχείο στο οποίο θα κάνει το post, δηλαδή σαν action, έχει το start.php. Οι κλάσεις που περιέχονται στα html elements αφορούν το styling του bootstrap template.

```
<script>
    function ulogin(){
        var ar = {
        username: $('#emailInput').val(),
        password: $('#passwordInput').val()
    };
        $.ajax( {type: "POST",url: "http://95.216.140.67:3000/users/login",
            data: ar,
            dataType: "json",
            success: function(result){
                console.log(result);
                if(result.message=='Auth successful') {
                    window.location.replace("http://95.216.140.67/eventap/index.html");
                    console.log('logged in');

                }
            },
            error: function(xhr){
                alert('Έχετε εισάγει λάθος στοιχεία');
            }
        });
    }
</script>
```

Εικόνα 54: Στιγμιότυπο της μεθόδου σύνδεσης χρήστη στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων

Η συγκεκριμένη μέθοδος είναι υπεύθυνη ώστε να πραγματοποιηθεί ή όχι το login. Ο τρόπος με τον οποίο δουλεύει είναι με μια κλήση ajax. Παρακάτω θα αναλύσουμε την υλοποίηση της. Καταρχάς σε έναν πίνακα με όνομα ar αποθηκεύονται το username και το password. Στη συνέχεια γίνεται μια POST κλήση στο endpoint που έχουμε δημιουργήσει στο API και είναι υπεύθυνο για το login. Τα data που αποστέλλονται είναι ο πίνακας ar και αποστέλλονται σε μορφή json. Παρακάτω και εφόσον η κλήση είναι επιτυχής, αν το μήνυμα που επιστρέφει είναι το 'Auth successful' τότε ανακατευθύνουμε τον χρήστη στην σελίδα index.html και έχει γίνει login. Διαφορετικά εμφανίζουμε το μήνυμα 'Έχετε εισάγει λάθος στοιχεία'.

6.4 index.html

Η index.html είναι η σελίδα που περιέχει όλο το περιεχόμενο που προβάλλεται σε κάποιον διαχειριστή που έχει κάνει επιτυχώς σύνδεση. Με το που γίνει επιτυχώς η σύνδεση, τότε εκτελείται μια κλήση AJAX προς το API και φέρνει τις εκδηλώσεις του συγκεκριμένου διοργανωτή. Παρακάτω θα αναλύσουμε τον κώδικα.

```
<script>
    events = '';
    var usrid = "@@usrid@@";
    bookings = [];

    function showEvents(){
          var ar = {};
            $.ajax( {type: "GET",url: "http://95.216.140.67:3000/events/org/"+usrid+"",
            //$.ajax(    {type: "GET",url: showeventsurl,
            data: ar,
            dataType: "json",
            success: function(result){

                console.log(result);
                fillTable(result);
                events = result;
            }
            ,
            error: function(xhr){
              alert('Request Status: ' + xhr.status + ' Status Text: ' + xhr.statusText + ' ' + xhr.responseText);
            }

            });
        }
```

Εικόνα 55: Στιγμιότυπο της συνάρτησης showEvents στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων

Καταρχάς αρχικοποιούμε μια μεταβλητή events στην οποία θα αποθηκευτούν οι εκδηλώσεις του συγκεκριμένου διοργανωτή, αν και εφόσον υπάρχουν. Στην επόμενη σειρά υπάρχει μια μεταβλητή usrid η οποία προσδιορίζει το id του χρήστη που έκανε σύνδεση. Όταν γίνεται η επιτυχώς σύνδεση έχω ορίσει παραπάνω να γίνεται αντικατάσταση του @@usrid@@ με το id του συγκεκριμένου χρήστη. Παρακάτω αρχικοποιώ και έναν πίνακα με όνομα bookings ώστε εκεί να αποθηκευτούν οι κρατήσεις για τις εκδηλώσεις. Έπειτα ορίζουμε μια μέθοδο showEvents στην οποία θα κάνουμε την κλήση. Το πρώτο πράγμα που πρέπει να κάνουμε είναι να ορίσουμε μια μεταβλητή για παράδειγμα με όνομα ar, η οποία θα πάρει τα δεδομένα που θα στείλουμε στην κλήση. Στην προκειμένη περίπτωση δεν υπάρχει κάτι να στειλουμε, παρά μόνο το usrid στο url συνεπώς το ar θα μείνει κενό. Στην επόμενη γραμμή κάνουμε μια AJAX κλήση τύπου GET στο συγκεκριμένο url, το οποίο επιστρέφει τις εκδηλώσεις ενός συγκεκριμένου διοργανωτή με το συγκεκριμένο usrid. Έπειτα ορίζουμε ότι τα data είναι το ar, ότι ο τύπος δεδομένων (datatype) είναι JSON και αν όλα πάνε καλά στην κλήση τότε μπαίνει στο success και τυπώνουμε στην κονσόλα το αποτέλεσμα, αλλά και γεμίζουμε τη μεταβλητή events με το αποτέλεσμα, όπως επίσης καλούμε και την fillTable η οποία θα γεμίσει τον πίνακα των εκδηλώσεων με τις εκδηλώσεις. Αν κάτι δεν πάει καλά τότε μπαίνει στο error και μας επιστρέφει το σφάλμα. Παρακάτω θα αναλύσουμε τι κάνει η fillTable.

Η μέθοδος fillTable παίρνει μια παράμετρο την eventsArray η οποία είναι οι εκδηλώσεις που έχουν επιστραφεί από την κλήση. Καταρχάς αδειάζουμε το HTML περιεχόμενο της evbody με την εντολή document.getElementById('evbody').innerHTML = '' και ορίζουμε μια μεταβλητή tableRows. Στη συνέχεια εκτελούμε ένα for loop αναλόγως με το πλήθος των εκδηλώσεων και δημιουργούμε το table με την ημερομηνία, τον τίτλο, την περιγραφή, την τοποθεσία και την χωρητικότητα.

```
function fillTable(eventsArray){
    document.getElementById("evbody").innerHTML = "";
    var tableRows = "" ;

    for(var i=0;i<eventsArray.length;i++){
        tableRows += "<tr>";
        tableRows +=    "<td><center>"+formatDate(eventsArray[i].date)+"<center></td>";
        tableRows +=    "<td><center>"+eventsArray[i].title+"<center></td>";
        tableRows +=    "<td><center>"+eventsArray[i].descr+"<center></td>";
        tableRows +=    "<td><center>"+eventsArray[i].venue+"<center></td>";
        tableRows +=    "<td><center> <span id='capacitycompleted"+[i]+"'>0</span>/"+eventsArray[i].capacity+"<center></td>";
        tableRows +=    `<td id=ev"+i+">
                            <button onclick='fillEditEventModal("+i+")' data-toggle='modal' data-target='#editEventModal' type='
                            <button data-toggle='modal' data-target='#showReservationsModal' id=`+eventsArray[i]._id+` onclick="
                        </td>`;
        tableRows += "</tr>";
    }
    document.getElementById("evbody").innerHTML = tableRows;
}
```

Εικόνα 56: Στιγμιότυπο της συνάρτησης fillTable στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων

Τέλος αντικαθιστούμε το HTML περιεχόμενο του evbody με την μεταβλητή tableRows, δηλαδή με τις εκδηλώσεις που λάβαμε από την κλήση. Στη συνέχεια και εφόσον υπάρχουν εκδηλώσεις, σε κάθε εκδήλωση δημιουργείται ένα κουμπί “Επεξεργασία Εκδήλωσης” όπου ο διοργανωτής μπορεί να επεξεργαστεί κάποια πληροφορία από την εκδήλωση. Πρώτα όμως πρέπει να “γεμίσουμε” με πληροφορίες το modal της κάθε εκδήλωσης. Αυτό κάνει η παρακάτω μέθοδος fillEditEventModal.

```
function fillEditEventModal(k){
    console.log(events[k]);
    document.getElementById('editEventTitle').value = events[k].title;
    document.getElementById('editEventDate').value = formatDate(events[k].date);
    document.getElementById('editEventDescr').value = events[k].descr;
    document.getElementById('editEventLocation').value = events[k].venue;
    document.getElementById('editEventPrice').value = events[k].price;
    document.getElementById('editEventImg').value = events[k].img;
}
```

Εικόνα 57: Στιγμιότυπο της μεθόδου fillEditEventModal στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων

Στη συνέχεια υπάρχει η μέθοδος fillReservations η οποία όπως και η showEvents, μέσω μιας AJAX κλήσης φέρνει όλες τις κρατήσεις που υπάρχουν για μια συγκεκριμένη εκδήλωση. Ο τρόπος υλοποίησης είναι ίδιος με παραπάνω, συνεπώς δεν υπάρχει ανάγκη για περαιτέρω ανάλυση.

```
function fillReservations(k,id){
    var ar = {};
    $.ajax( {type: "GET",url: "http://95.216.140.67:3000/enrollments/evt/"+id+"",
    data: ar,
    dataType: "json",
    success: function(result){
        console.log("book:" +JSON.stringify(result));
        bookings = result;
        s = '';
        document.getElementById('bookingsbody').innerHTML ='';
        document.getElementById('capacitycompleted'+k).innerHTML = bookings.length;
        for(var i=0;i<bookings.length;i++){
            s+= '<tr>';
            s+= '<th class=aa scope="row">'+(i+1)+'</th>';
            s+= '<td class=fullname>'+bookings[i].fullname+'</td>';
            s+= '<td class=fullname>'+bookings[i].tickets+'</td>';
            s+= '<td class=fullname>'+bookings[i].totalPrice+'€</td>';
            s+= '</tr>';
        }
        document.getElementById('bookingsbody').innerHTML += s;
    }
    ,
    error: function(xhr){
      alert('Request Status: ' + xhr.status + ' Status Text: ' + xhr.statusText + ' ' + xhr.responseText);
    }

    });
}
```

Εικόνα 58: Στιγμιότυπο της συνάρτησης fillReservations στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων

Αξίζει να γίνει αναφορά σε μερικές βοηθητικές μεθόδους. Για παράδειγμα υπάρχει η μέθοδος formatDate η οποία διαβάζει μια ημερομηνία με τη μορφή που υπάρχει στη βάση δεδομένων (πχ 2021-05-13) και τη μετατρέπει σε 13/05/2021. Τη χρησιμοποίησα μόνο και μόνο για να υπάρχει μια πιο ωραία εικόνα προς τον χρήστη.

```
function formatDate(date) {
        var d = new Date(date),
            month = '' + (d.getMonth() + 1),
            day = '' + d.getDate(),
            year = d.getFullYear();

        if (month.length < 2) month = '0' + month;
        if (day.length < 2) day = '0' + day;
        return [day, month, year].join('/');
}
```

Εικόνα 59: Στιγμιότυπο της συνάρτησης formatDate στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων

Έπειτα υπάρχει η μέθοδος insertEvt με την οποία ο διοργανωτής μπορεί να δημιουργήσει μια εκδήλωση. Στη συγκεκριμένη περίπτωση υπάρχουν δεδομένα που αποστέλλονται μέσω της AJAX κλήσης. Τα δεδομένα που αποστέλλονται τα βλέπουμε παρακάτω και είναι τα:

- Τίτλος εκδήλωσης
- Περιγραφή εκδήλωσης
- Ημερομηνία εκδήλωσης
- id του διοργανωτή
- Τοποθεσία εκδήλωσης
- Χωρητικότητα εκδήλωσης
- Τιμή εισιτηρίου εκδήλωσης
- url φωτογραφίας εκδήλωσης

```
function insertEvt(){
    var ar = {};
    ar.title = document.getElementById('evttitle').value;
    ar.descr = document.getElementById('evtdescr').value;
    ar.date = document.getElementById('evtdate').value;
    ar.organizerId = document.getElementById('usrid').getAttribute('usrid');
    ar.venue = document.getElementById('evtlocation').value;
    ar.capacity = document.getElementById('evtcapacity').value;
    ar.price = document.getElementById('evtprice').value;
    ar.img = document.getElementById('evtimgurl').value;
    $.ajax( {type: "POST",url: "http://95.216.140.67:3000/events",
    data: ar,
    dataType: "json",
    success: function(result){
        alert("Δημιουργήθηκε επιτυχώς");
    },
    error: function(xhr){
      alert('Request Status: ' + xhr.status + ' Status Text: ' + xhr.statusText + ' ' + xhr.responseText);
    }
    });
}
```

Εικόνα 60: Στιγμιότυπο της συνάρτησης insertEvt στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων

Εφόσον η εκδήλωση δημιουργηθεί επιτυχώς μας επιστρέφει το μήνυμα “Δημιουργήθηκε επιτυχώς”. Αν κάτι δεν πάει καλά, πηγαίνει στο error και μας εμφανίζεται με ένα alert το αντίστοιχο σφάλμα.

Παρακάτω αξίζει να δούμε το modal το οποίο “σηκώνεται” και ο διοργανωτής έχει τη δυνατότητα να συμπληρώσει τα στοιχεία της νέας εκδήλωσης ώστε να τη δημιουργήσει. Επί της ουσίας είναι ένα μεγάλο div στο οποίο περικλύονται μικρότερα divs τα οποία το καθένα με τη σειρά του περιέχει ένα από τα απαραίτητα πεδία. Βάση του id που έχει (στην συγκεκριμένη περίπτωση το newEventModal) κάνοντας κλικ στο κουμπί “Δημιουργία Εκδήλωσης” ανοίγει αυτό το div. Παρακάτω παρατίθεται αναλυτικά ο κώδικας.

```
<!--Modal Νέα Εκδήλωση-->
<div class="modal fade" id="newEventModal" tabindex="-1" role="dialog" aria-labelledby="newEventModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title">Νέα εκδήλωση</h5>
          <button type="button" class="close" data-dismiss="modal" aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
        <div class="modal-body">
            <div class="form-group">
                <label class="control-label">Τίτλος</label>
                <div>
                    <input type="text" class="form-control input-lg" name="title" id="evttitle">
                </div>
            </div>
            <div class="form-group">
                <label class="control-label">Ημερομηνία</label>
                <div>
                    <input type="date" class="form-control input-lg" name="date" id="evtdate">
                </div>
            </div>
            <div class="form-group">
                <label for="exampleFormControlTextarea1">Περιγραφή</label>
                <textarea class="form-control" id="evtdescr" rows="3"></textarea>
            </div>
            <div class="form-group">
                <label class="control-label">Τοποθεσία</label>
                <div>
                    <input type="text" class="form-control input-lg" name="location" id="evtlocation">
                </div>
            </div>
            <div class="form-group">
                <label class="control-label">Τιμή εισιτηρίου</label>
                <div>
                    <input type="number" class="form-control input-lg" name="price" id="evtprice">
                </div>
            </div>
            <div class="form-group">
                <label class="control-label">Χωρητικότητα</label>
                <div>
                    <input type="number" class="form-control input-lg" name="capacity" id="evtcapacity">
                </div>
```

Εικόνα 61: Στιγμιότυπο του modal νέας εκδήλωσης στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων 1 από 2

```
                </div>
                <div class="form-group">
                    <label class="control-label">URL Εικόνας</label>
                    <div>
                        <input type="text" class="form-control input-lg" name="img_url" id="evtimgurl">
                    </div>
                </div>
            </div>
            <div class="modal-footer">
              <button type="button" class="btn btn-success" onclick="insertEvt()">Καταχώρηση</button>
              <button type="button" class="btn btn-danger" data-dismiss="modal">Άκυρο</button>
            </div>
          </div>
        </div>
    </div>
<!--Modal Νέα Εκδήλωση ΤΕΛΟΣ-->
```

Εικόνα 62: Στιγμιότυπο του modal νέας εκδήλωσης στο αρχείο index.html της εφαρμογής διαχείρισης εκδηλώσεων 2 από 2

6.5 Επίλογος

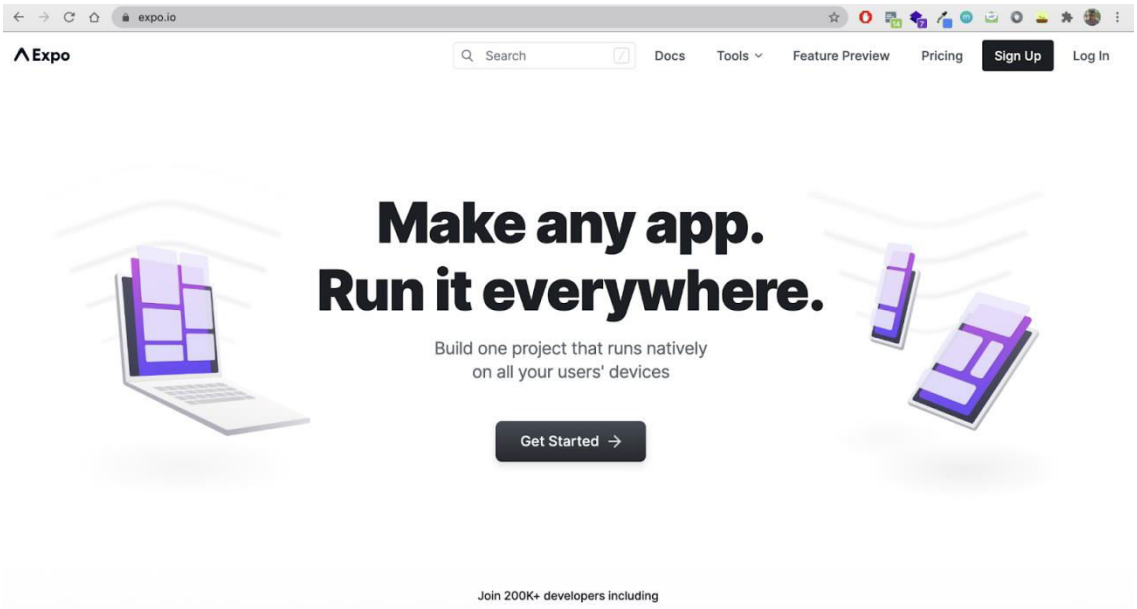
Παραπάνω αναλύθηκαν οι βασικές λειτουργίες του web application διαχείρισης εκδηλώσεων. Χάρη στο AJAX και την PHP καταφέραμε εύκολα και γρήγορα να υλοποιήσουμε το παραπάνω. Παρακάτω θα δούμε αναλυτικά την ανάπτυξη της mobile εφαρμογής για τους απλούς χρήστες.

Κεφάλαιο 7ο: Ανάπτυξη mobile εφαρμογής

7.1 Εισαγωγη

Στην παρούσα ενότητα θα αναλύσουμε την ανάπτυξη της mobile εφαρμογής προβολής και κράτησης εκδηλώσεων για τους χρήστες. Όπως είναι ήδη γνωστό από προηγούμενη ενότητα η εφαρμογή έχει αναπτυχθεί με τη βιβλιοθήκη React Native. Επίσης βασικό εργαλείο για την ανάπτυξη της ήταν το expo.io το οποίο προσφέρει μερικές έτοιμες λύσεις σε διάφορα προβλήματα που στόχο έχουν την γρηγορότερη και αποδοτικότερη ανάπτυξη μιας εφαρμογής.

7.2 Expo.io



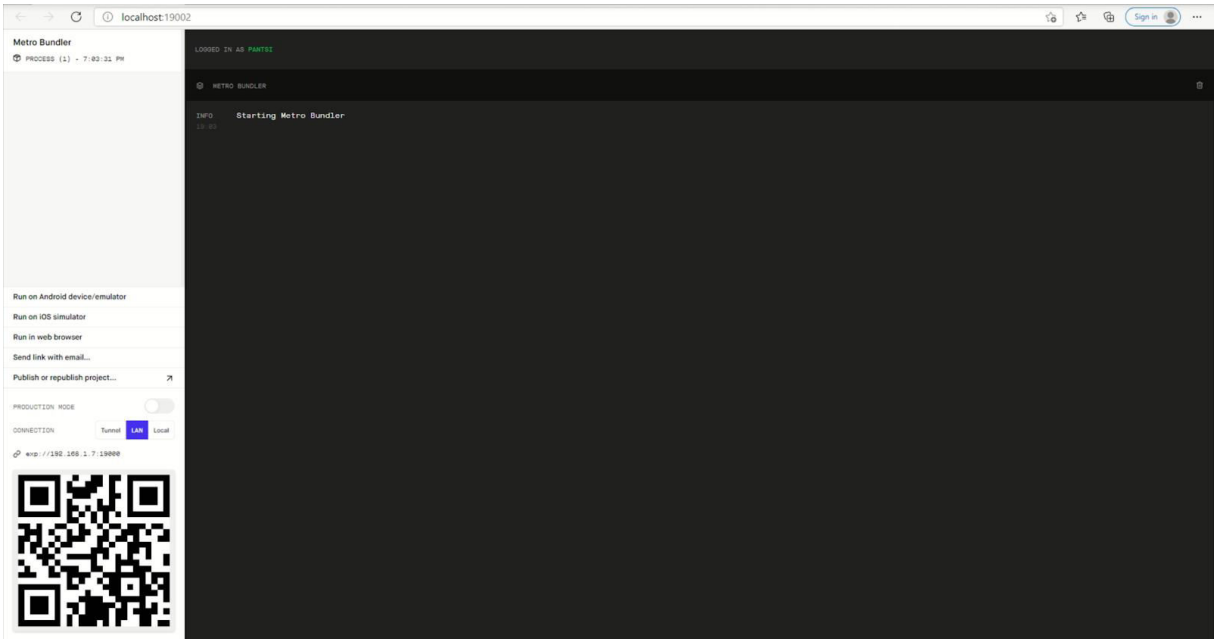
Εικόνα 63: Στιγμιότυπο της ιστοσελίδας expo.io [5]

Δύο από τα τεράστια πλεονεκτήματα που προσφέρει η expo είναι ότι κατά την ανάπτυξη μιας εφαρμογής, μπορεί ο προγραμματιστής να βλέπει άμεσα τις αλλαγές εγκαθιστώντας την εφαρμογή expo για κινητά. Επί της ουσίας, αφότου ο χρήστης έχει εγκαταστήσει την εφαρμογή, σκανάροντας το QR code που δημιουργείται εκκινεί η εφαρμογή και μπορεί να παρατηρεί και ταυτόχρονα να αναπτύσει. Το δεύτερο μεγάλο πλεονέκτημα του expo είναι ότι μπορεί πολύ εύκολα ο προγραμματιστής να προβεί σε build μέσα από servers του expo και να γλιτώσει πολύτιμο χρόνο. Εφόσον έχουμε κάνει την απαραίτητη εισαγωγή στο expo, ας ξεκινήσουμε να δούμε πως εγκαθίσταται στον υπολογιστή.

7.2.1 Εγκατάσταση Expo και εκκίνηση εφαρμογής

Το πρώτο βήμα που πρέπει να κάνουμε είναι να βεβαιωθούμε ότι έχουμε εγκατεστημένη την node.js στον υπολογιστή. Στη συνέχεια πρέπει να ανοίξουμε το command line και να πληκτρολογήσουμε τις εξής εντολές:

- **npm install --global expo-cli** (η συγκεκριμένη εντολής εγκαθιστά global - σε όλο τον υπολογιστή το expo cli)
- **expo init my-app** (η συγκεκριμένη εντολή αρχικοποιεί και εγκαθιστά στον φάκελο my-app τα αρχεία που χρειάζεται η εφαρμογή για να εκκινήσει)
- **cd my-app** (με τη συγκεκριμένη εντολή μεταφερόμαστε εντός του φακέλου my-app)
- **expo start** (με τη συγκεκριμένη εντολή εκκινεί η εφαρμογή και ανοίγει αυτόματα η παρακάτω σελίδα όπου σκανάρουμε το qr code στο κάτω αριστερό μέρος κι έτσι αποκτούμε πρόσβαση από το κινητό.



Εικόνα 64: Στιγμιότυπο του παραθύρου του metro bundler από το expo.io

Σε αυτό το σημείο αξίζει να αναφερθεί πως η ανάπτυξη της εφαρμογής γίνεται localhost, δηλαδή τοπικά στον υπολογιστή. Μπορούμε να παρατηρήσουμε πάνω στην μπάρα του url ότι τρέχει localhost στην πόρτα 19002. Οι δυνατότητες εκτέλεσης είναι οι εξής: είτε το κινητό να είναι συνδεδεμένο στο ίδιο δίκτυο με τον υπολογιστή που γίνεται η ανάπτυξη της εφαρμογής, είτε να είναι συνδεδεμένο το κινητό με κάποιο usb στον υπολογιστή, είτε μέσω tunnel, που αυτό συνεπάγεται πως μπορεί κάποιος να τρέξει την εφαρμογή σε οποιοδήποτε σημείο του κόσμου κι αν είναι. Η βασική διαφορά των τριών είναι η ταχύτητα εκτέλεσης και ανανέωσης της εφαρμογής. Παρακάτω θα αναφερθούμε στα βασικά αρχεία της εφαρμογής.

7.3 Βασικά αρχεία εφαρμογής

Στη συγκεκριμένη ενότητα θα δούμε και θα αναλύσουμε επιγραμματικά τα βασικά αρχεία της εφαρμογής. Με τη δημιουργία του project το expo δημιουργεί αυτόματα 3 φακέλους οι οποίοι περιέχουν κρυφά αρχεία και είναι οι .expo , ο .expo-shared και ο .vscode. Έπειτα υπάρχει ο φάκελος assets, ο οποίος περιέχει κάποιες εικόνες, όπως για παράδειγμα την εικόνα του λογοτύπου της οθόνης login. Στη συνέχεια υπάρχει ο φάκελος node_modules που περιέχει τα dependencies που θα δούμε παρακάτω. Επόμενος είναι ο φάκελος src που περιλαμβάνει τον κώδικα της εφαρμογής. Μέσα σε αυτό τον φάκελο υπάρχουν οι εξής υποφάκελοι:

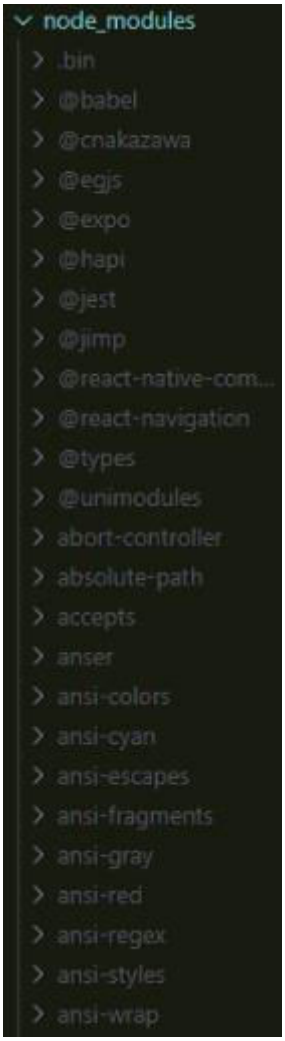
- alerts: Περιέχει τα alerts, όπως για παράδειγμα, το alert για την επιτυχή κράτηση
- api: Περιέχει το αρχείο που αρχικοποιεί τη σύνδεση με το API Service
- components: Περιέχει αρχεία τα οποία είναι επαναχρησιμοποιούμενα στην εφαρμογή
- screens: Περιέχει τις οθόνες της εφαρμογής

Εικόνα 65: Στιγμιότυπο της δομής των αρχείων της εφαρμογής για κινητά

Παρακάτω θα αναλύσουμε λίγο περισσότερο τα dependencies.

7.4 Dependencies

Όπως προαναφέραμε στον φάκελο node_modules υπάρχουν όλα τα dependencies. Αυτό σημαίνει πως όποια τρίτη επέκταση χρησιμοποιήσουμε βρίσκεται και ζει στον φάκελο node_modules. Για παράδειγμα στο κάτω μέρος της οθόνης της εφαρμογής υπάρχει ένα μενού. Για να επιτευχθεί αυτό χρειάστηκε να προστεθεί το dependencie με όνομα react-navigation. Ο τρόπος με τον οποίο εγκαθίστανται τέτοιου είδους επεκτάσεις είναι με το npm (node package manager). Το μόνο που χρειάζεται είναι να γράψουμε στην κονσόλα npm install -g ….. και το όνομα του ανάλογου dependencie και στη συνέχεια να κάνουμε επανεκκίνηση στην εφαρμογή.

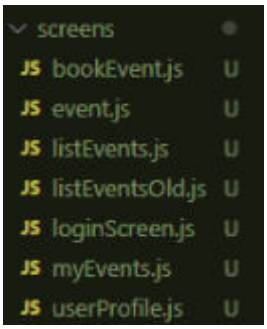


Εικόνα 66: Στιγμιότυπο βασικών dependencies της εφαρμογής για κινητά

Το ιδανικό σενάριο είναι να χρησιμοποιούνται όσο λιγότερα dependencies γίνεται ώστε να είναι πιο ελαφριά και πιο ευέλικτη η εφαρμογή. Βέβαια πέρα από αυτό υπάρχει ενδεχόμενο - μικρό αλλά υπάρχει - δύο επεκτάσεις να "χτυπάνε" μεταξύ τους. Δηλαδή για παράδειγμα να εκτελούν κάποιες κοινές μεταξύ τους διεργασίες και να κάνει overwrite η μια την άλλη. Τέλος θα πρέπει να προσέχουμε αρκετά οι επεκτάσεις που χρησιμοποιούμε να έχουν ενημερωθεί πρόσφατα ώστε να είμαστε σίγουροι πως δεν υπάρχει κάποιο κενό ασφαλείας. Εφόσον έχουμε αναφερθεί και στα dependencies έχει έρθει η στιγμή να δούμε τον κώδικα των οθονών της εφαρμογής.

7.5 Κώδικας οθονών εφαρμογής

Σε αυτή την ενότητα θα αναλύσουμε τον κώδικα που περιέχουν τα αρχεία των οθονών της εφαρμογής. Όπως έχουμε αναφέρει παραπάνω τα αρχεία των οθονών βρίσκονται στον φάκελο screens και είναι αυτά που παρουσιάζονται στο παρακάτω screenshot. Παρακάτω θα αναλύσουμε το καθένα ξεχωριστά



Εικόνα 67: Στιγμιότυπο των αρχείων των οθονών της εφαρμογής για κινητά

7.5.1 App.js

Το App.js είναι το βασικό αρχείο και ο κορμός της εφαρμογής. Σε αυτό συγκεντρώνονται όλες οι οθόνες ώστε να υλοποιηθεί ορθά το routing, δηλαδή όταν ο χρήστης κάνει κλικ από τη μια οθόνη για να πάει στην άλλη. Επιπλέον πέρα από τις οθόνες συμπεριλαμβάνονται και κάποιες επιπλέον βιβλιοθήκες, για παράδειγμα η Ionicons με την οποία μπορούμε να προσθέσουμε icons.

```
import * as React from 'react';
import { View, Text,Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import listEvents from './src/screens/listEvents';
import loginScreen from './src/screens/loginScreen';
import event from './src/screens/event';
import Ionicons from 'react-native-vector-icons/Ionicons';
import bookEvent from './src/screens/bookEvent';
import myEvents from './src/screens/myEvents';
import userProfile from './src/screens/userProfile';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title = "Events"
        onPress={()=> navigation.navigate('Events')}
      />
    </View>
  );
}
```

Εικόνα 68: Στιγμιότυπο κώδικα του αρχείου App.js 1/3

Όσον αφορά τον παραπάνω κώδικα, αυτά τα οποία γίνονται import είναι τα εξής:

- * as React , δηλαδή ολόκληρη η React library
- View,Text,Button δηλαδή τα αντικείμενα αναπαράστασης μιας οθόνης, κειμένου και κουμπιού της react native
- NavigationContainer το οποίο ευθύνεται για το routing
- createStackNavigator το οποίο επίσης ευθύνεται για το routing
- listEvents, loginScreen, event, bookEvent, myEvents, userProfile οθόνες
- createBottomTabNavigator το οποίο είναι υπεύθυνο για το menu στο κάτω μέρος της οθόνης

Στη συνέχεια υπάρχει η συνάρτηση HomeScreen η οποία παίρνει ως παράμετρο το navigation. Αυτή η συνάρτηση προβάλλει τις εκδηλώσεις και με το navigation, ο χρήστης κάνοντας tap σε μία εκδήλωση μεταφέρεται στη συγκεκριμένη εκδήλωση.

```
const Stack = createStackNavigator();
const Tab = createBottomTabNavigator();

function EventsMenu() {
  return (
    <Tab.Navigator
    screenOptions={({ route }) => ({
      tabBarIcon: ({ focused, size }) => {
        let iconName;

        if (route.name === 'Εκδηλώσεις') {
          iconName = focused
            ? 'ios-calendar'
            : 'ios-calendar-outline';
        } else if (route.name === 'Προφίλ') {
          iconName = focused ? 'ios-person' : 'ios-person-outline';
        }
        return <Ionicons name={iconName} size={size} color='grey' />;
      }
    }
    )}
    >
      <Tab.Screen name="Εκδηλώσεις" component={listEvents} />
      <Tab.Screen name="Προφίλ" component={userProfile} />
    </Tab.Navigator>
  );
}
```

Εικόνα 69: Στιγμιότυπο κώδικα του αρχείου App.js 2/3

Έπειτα αρχικοποιείται το Stack και το Tab έτσι ώστε να μπορέσουμε να δημιουργήσουμε στο κάτω μέρος της οθόνης το menu. Στη συνέχεια προχώραμε στην υλοποίηση του με τη συνάρτηση EventsMenu(). Επί της ουσίας η συγκεκριμένη συνάρτηση δημιουργεί ένα Tab Navigator και αν ο χρήστης επιλέξει “Εκδηλώσεις” τότε τον μεταφέρει στις εκδηλώσεις και αλλάζει το αντίστοιχο χρώμα στο μενού. Αν επίλεξει το “Προφίλ” τότε συβαίνει κάτι αντίστοιχο. Στο κάτω μέρος δημιουργούνται 2 elements τύπου Tab.Screen και αυτά είναι που προβάλλονται στον χρήστη και με το component που περιέχουν σαν παράμετρο, γνωρίζει το λογισμικό που θα παραπέμψει τον χρήστη όταν κάνει tap.

```
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Login"
          component={loginScreen}
          options = {{headerShown: false}}
        />
        <Stack.Screen
          name="Eventap"
          component={EventsMenu}
          options = {{headerLeft: false}}
        />
        <Stack.Screen
          name="Event"
          component={event}
        />
        <Stack.Screen
          name="Κράτηση"
          component={bookEvent}
        />
        <Stack.Screen
          name="Οι κρατήσεις μου"
          component={myEvents}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );

}

export default App;
```

Εικόνα 70: Στιγμιότυπο κώδικα του αρχείου App.js 3/3

Τέλος, στο συγκεκριμένο αρχείο υπάρχει η βασική συνάρτηση App() η οποία είναι υπεύθυνη για την εκτέλεση της εφαρμογής. Μέσα σε αυτή δημιουργείται το λεγόμενο NavigationContainer όπου ορίζουμε μικρότερα components τύπου Stack.Screen και ορίζουν το όνομα, τα components και τα options της κάθε οθόνης. Αναλυτικά έχουμε τα εξής Stack screens:

- Login: ως όνομα έχει το Login, όταν καλείται χρησιμοποιεί το loginScreen (loginScreen.js όπως έχουμε ορίσει παραπάνω) και έχουμε ορίσει να μην έχει κάποια κεφαλίδα.
- EventAp: ως όνομα έχει το Eventap, όταν καλείται χρησιμοποιείται το EventsMenu (eventsMenu.js) κα έχουμε ορίσει να μην έχει κεφαλίδα στα αριστερά, δηλαδή το βελάκι για να πάει κάποιος πίσω.
- Event: ως όνομα έχει το Event και όταν καλείται χρησιμοποιεί το event (event.js όπως έχουμε ορίσει παραπάνω) και αφορά την οθόνη μιας εκδήλωσης
- Κράτηση: ως όνομα έχει το Κράτηση και όταν καλείται χρησιμοποιεί το booking (booking.js όπως έχουμε ορίσει παραπάνω) και αφορά την οθόνη κράτησης μιας εκδήλωσης
- Οι κρατήσεις μου: ως όνομα έχει το 'Οι κρατήσεις μου' και όταν καλείται χρησιμοποιεί το myEvents (myEvents.js όπως έχουμε ορίσει παραπάνω) και αφορά τις εκδηλώσεις ενός χρήστη.

Τα παραπάνω είναι όσα περιέχονται στο βασικότερο αρχείο της εφαρμογής. Παρακάτω θα αναλύσουμε την κάθε οθόνη και τον τρόπο που έχει αναπτυχθεί ώστε να υπάρχει καλή εμπειρία χρήστη.

7.5.2 Login Screen

Στη συγκεκριμένη ενότητα θα αναλύσουμε τον κώδικα της οθόνης σύνδεσης ενός χρήστη. Όπως και παραπάνω στο αρχείο App.js έτσι και εδώ γίνεται import κάποιων αρχείων ή dependencies ή βιβλιοθηκών. Συγκεκριμένα από τη React επιπλέον χρησιμοποιούμε το useState και το useEffect. Τα συγκεκριμένα συγκαταλέγονται στα λεγόμενα hooks και είναι υπεύθυνα να εκτελέσουν κάποιο κομμάτι κώδικα εφόσον γίνει render. Επιπλέον χρησιμοποιούμε τα εξής από την React Native.

- StyleSheet: Μπορούμε να χρησιμοποιήσουμε css μέσα στην εφαρμογή μας
- Text: Μπορούμε να έχουμε πεδία Text
- View: Μπορούμε να έχουμε το View, ώστε να εμφανίσουμε την οθόνη
- Image: Μπορούμε να χρησιμοποιήσουμε εικόνα
- TextInput: Μπορούμε να έχουμε πεδίο κειμένου ώστε να γράψει ο χρήστης
- TouchableOpacity: Μπορούμε να έχουμε κείμενο ή εικόνα και να κάνει κλικ ο χρήστης

```
import React, { useState, useEffect } from 'react';
import {StyleSheet, Text, View,Image,TextInput,TouchableOpacity} from 'react-native';
import api from '../api/api';
import * as SecureStore from 'expo-secure-store';
import WrongCredsAlert from '../alerts/WrongCredsAlert';


async function save(key,value) {
    await SecureStore.setItemAsync(key,value);
}

const loginScreen = (props) => {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");

function clearFields() {
    setEmail("");
    setPassword("");
}

const doLogin = async () => {
    let loginResponse;
    api.post(`/users/login`,{
        username: email.toLowerCase(),
        password: password
    })
        .then(async(response)=>{
            if(response.status === 200){
                props.navigation.navigate('Eventap')
                save('username',email.toLowerCase())
                save('password',password)
                save('uid',response.data.userId)
            }
        })
        .catch(e=>{
            if(e.response.status === 401){
                WrongCredsAlert()
                console.log('wrong credentials')
            }
        })
}
```

Εικόνα 71: Στιγμιότυπο κώδικα του αρχείου LoginScreen.js 1/2

Στη συνέχεια κάνουμε import το Api, το οποίο θα αναλύσουμε στη συνέχεια. Επί της ουσίας περιέχει κάποιες εντολές που ορίζουν το URL στο οποίο θα κάνουμε τις κλήσεις. Έπειτα χρησιμοποιούμε το SecureStore του Expo, το οποίο είναι μια ενσωματωμένη βιβλιοθήκη η οποία μας προσφέρει τη δυνατότητα να αποθηκεύουμε κρυπτογραφημένες πληροφορίες τοπικά στη συσκευή του χρήστη. Τέλος χρησιμοποιούμε ένα Alert το οποίο έχουμε δημιουργήσει και αφορά το μήνυμα που θα εμφανιστεί αν κάποιος χρήστης βάλει λάθος τα στοιχεία σύνδεσης του.

Παρακάτω έχουμε τη συνάρτηση save() με την οποία αποθηκεύουμε δεδομένα τοπικά στη συσκευή. Στη συνέχεια ορίζεται η συνάρτηση loginScreen με παράμετρο την props η οποία αφορά ολόκληρη την οθόνη. Μέσα σε αυτή με την useState διαχειριζόμαστε τις πληροφορίες που εισάγει ο χρήστης (όνομα χρήστη και κωδικό) και με την doLogin η οποία είναι μια ασύγχρονη συνάρτηση κάνουμε ένα POST request στο /users/login και πραγματοποιείται η αυθεντικοποίηση του χρήστη. Αν η αυθεντικοποίηση είναι επιτυχής τότε μέσω του SecureStore αποθηκεύονται το username, το password και το uid του χρήστη τοπικά στη συσκευή και ο χρήστης ανακατευθύνεται στην οθόνη των εκδηλώσεων. Αν συμβεί κάποιο σφάλμα εμφανίζεται το ανάλογο alert. Στο παρακάτω στιγμιότυπο κώδικα μπορούμε να δούμε αναλυτικά τον κώδικα του UI της συγκεκριμένης οθόνης.

```
return (
    <View style={styles.container}>
        <Image source={require('../../assets/eventaplogo.png')} style={styles.eventaplogoStyle}></Image>
        <TextInput
            label="E-mail"
            value={email}
            placeholder="E-mail"
            placeholderTextColor="#fff"
            style={styles.inputs}
            onChangeText={(text) => setEmail(text)}
        />
        <TextInput
            label="Password"
            value={password}
            placeholder="Κωδικός"
            placeholderTextColor="#fff"
            textContentType="password"
            secureTextEntry={true}
            style={styles.inputs}
            onChangeText={(text) => setPassword(text)}
        />

        <TouchableOpacity
            style={{backgroundColor:'#fff',padding:10,marginBottom:5,borderRadius:5,width:320,alignSelf:'center',elevation:3,marginTop:10}}
            onPress = {doLogin}
        >
            <Text style={{fontSize:20,color:'#000',textAlign:'center'}}>Σύνδεση</Text>
        </TouchableOpacity>

        <TouchableOpacity
            style={{backgroundColor:'#fff',padding:10,marginBottom:5,borderRadius:5,width:250,alignSelf:'center',elevation:3,marginTop:10,backgroundColor:'#4e73df'}}
        >
            <Text style={{fontSize:15,color:'#fff',textAlign:'center'}}>Ξέχασα τον κωδικό μου</Text>
        </TouchableOpacity>


        <View style={styles.footer}>
            <TouchableOpacity title="NeedHelp" style={styles.bottomButton}>
                <Text style={styles.myText}>Χρειάζεσαι βοήθεια;</Text>
            </TouchableOpacity>
        </View>

    </View>
);
```

Εικόνα 72: Στιγμιότυπο κώδικα του αρχείου LoginScreen.js 2/2

Για να μπορέσει να δημιουργηθεί μια οθόνη, περικλύεται σε μια return() και στη συνέχεια μέσα σε ένα View. Μέσα στο View υπάρχει μια εικόνα και συγκεκριμένα το λογότυπο της εφαρμογής και παρακάτω δύο πεδία, ένα για το όνομα χρήστη και ένα για τον κωδικό. Έπειτα σε TouchableOpacity υπάρχει το κουμπί της σύνδεσης που όταν το επιλέξει ο χρήστης εκτελείται η συνάρτηση doLogin(). Στη συγκεκριμένη οθόνη χρησιμοποιείται inline styling, δηλαδή μέσα στο κάθε element παρατίθενται και οι εντολές για τη μορφοποίηση του. Έπειτα σε ένα ακόμη TouchableOpacity υπάρχει κουμπί ‘Ξέχασα τον κωδικό μου’, ανεπτυγμένο με την ίδια λογική. Τέλος υπάρχει το κουμπί ‘Χρειάζεσαι Βοήθεια’ και έτσι κλείνουν τα View και η return ώστε να μπορέσει να κάνει render η συγκεκριμένη οθόνη.

7.5.3 Events Screen

Στη συγκεκριμένη ενότητα θα αναλύσουμε τον κώδικα της οθόνης των εκδηλώσεων. Σχετικά με τα παραπάνω αυτό που προστίθεται επιπλέον είναι η βιβλιοθήκη moment η οποία χρησιμοποιείται για την πιο φιλική προς τον χρήστη παρουσίαση των ημερομηνιών. Για να είναι responsive η οθόνη, δηλάδη αναλόγως το κινητό και την ανάλυση του να προσαρμόζεται, υπολογίζουμε το μήκος και το πλάτος του κάθε αντικειμένου βάση δύο μεταβλητών που έχουμε ορίσει, των windowWidth και windowHeight, που όπως προδίδει το όνομα τους στη μια αποθηκεύεται το μήκος της οθόνης και στην άλλη το ύψος. Για να το πετύχουμε αυτό πρέπει να συμπεριληφθεί στο import το Dimensions ώστε να καταφέρουμε να γνωρίζουμε τις συκγκεριμένες πληροφορίες.

```
import React, {useState,useEffect} from 'react';
import {StyleSheet, Text, View,Image,TextInput,TouchableOpacity,FlatList,Dimensions } from 'react-native';
import api from '../api/api';
import * as SecureStore from 'expo-secure-store';
import moment from 'moment';


const windowWidth = Dimensions.get('window').width;
const windowHeight = Dimensions.get('window').height;

const listEvents = (props) => {
    const [result, setResult] = useState([]);
    const [username, setUsername] = useState([]);
    const [loading,setLoading] = useState(true);

    const getResult = async () => {
        const response = await api.get(`/events`);
        setResult(response.data);
        setLoading(false);
    };
    const getUsername = async () => {
        const response = await SecureStore.getItemAsync('username');
        setUsername(response);
    }
    useEffect(() => {
        getResult();
        getUsername();
    }, []);
```

Εικόνα 73: Στιγμιότυπο κώδικα του αρχείου listEvents.js 1/2

Στη συνέχεια υπάρχει η συνάρτηση listEvents η οποία ευθύνεται για όλη την οθόνη. Μέσα σε αυτήν αξίζει να αναφέρουμε ότι ορίζεται το loading, δηλαδή μέχρι να φορτώσουν οι εκδηλώσεις, υπάρχει μια μεταβλητή που ελέγχει πότε έχουν φορτώσει οι εκδηλώσεις και μέχρι αυτό να συμβεί υπάρχει ένας loading spinner που γυρνάει και έτσι βελτιώνεται το user experience καθώς ο χρήστης γνωρίζει ότι κάτι συμβαίνει. Έπειτα υπάρχει η συνάρτηση getUsername η οποία ανακτά το username από το SecureStore που το είχαμε αποθηκεύσει προηγουμένως ώστε να φορτώσει τις κατάλληλες εκδηλώσεις στον χρήστη. Και σε αυτή την οθόνη χρησιμοποιούμε hooks και συγκεκριμένα την useEffect. Στο παρακάτω στιγμιότυπο βλέπουμε πως εφόσον υπάρχουν εκδηλώσεις, δηλαδή εφόσον το result δεν είναι null τότε δημιουργείται το UI. Δύο σημαντικές παρατηρήσεις στο συγκεκριμένο σημείο είναι πως η εικόνα της κάθε εκδήλωσης έχει ύψος 150 pixels αλλά μήκος το 0.88 της οθόνης. Επιπλέον στο View στο οποίο περικλείονται οι πληροφορίες για την εκάστοτε εκδήλωση βλέπουμε ότι μέσα στη συνάρτηση moment ορίζουμε ως παράμετρο την ημερομηνία της εκδήλωσης και επιλέγουμε την ημερολογιακή μορφή που θα έχει, δηλαδή από ημερομηνία μέχρι ώρα, λεπτά και δευτερόλεπτα

```
if (result != null) {
    return (

        <View style={styles.container}>
            <FlatList
                keyExtractor={(event) => { return event._id }}
                data={result}
                renderItem={({ item }) => {
                return (
                    <View style={styles.eventCont}>
                        <TouchableOpacity
                            onPress={()=> props.navigation.navigate('Event',{
                                title: item.title,
                                eventid: item._id
                            })}
                        >
                        <Image
                            source = {{uri:`${item.img}`}}
                            style={{width: windowWidth*0.88, height: 150}}
                        />
                        <View style={styles.event}>

                            <View>
                                    <Text style={styles.title}>{item.title}</Text>
                                    <Text style={styles.descr}>{moment(item.date).format('DD/MM/yyyy')} {moment(item.date).utcOffset('GMT +00:00').format('HH:ss')}</Text>
                            </View>
                        </View>
                        </TouchableOpacity>
                    </View>
                );
                }
                }
            />
        </View>
    )
}
```

Εικόνα 74: Στιγμιότυπο κώδικα του αρχείου listEvents.js 2/2

Στη συνέχεια θα αναλύσουμε τον κώδικα μιας συγκεκριμένης εκδήλωσης, δηλαδή όταν ο χρήστης κάνει tap σε μια εκδήλωση από τη λίστα των εκδηλώσεων.

7.5.4 Event Screen

Στη συγκεκριμένη ενότητα θα αναλύσουμε τον κώδικα της οθόνης μιας εκδήλωσης. Ακολουθείται το ίδιο μοτίβο με παραπάνω, δηλαδή συμπεριλαμβάνονται η React και τα hooks, όπως επίσης και συγκεκριμένα elements της react native. Επιπλέον συμπεριλαμβάνεται το api για να υπάρξει η δυνατότητα να γίνει κλήση, η βιβλιοθήκη moment για να έχουμε μια διαφορετική μορφή στην ημερομηνία και τέλος το scroll view το οποίο δίνει τη δυνατότητα στον χρήστη να μπορεί να κάνει scroll σε μια συγκεκριμένη οθόνη. Το ScrollView πρόκειται για εξωτερική βιβλιοθήκη που έχει εγκατασταθεί μέσω του npm (node package manager).

Αρχικά με τη χρήση των hooks (useState) αρχικοποιείται η μεταβλητή tickets. Στη συνέχεια ορίζεται και η μέθοδος getTickets() η οποία εκτελείται κάθε φορά που ο χρήστης μπαίνει στη συγκεκριμένη οθόνη. Αυτό επιτυγχάνεται με τη χρήστη της useEffect. Έπειτα σε δύο μεταβλητές ορίζεται το μήκος και το ύψος της οθόνης της εκάστοτε συσκευής. Αυτό επιτυγχάνεται με το element με όνομα Dimensions που υπάρχει διαθέσιμο στη React Native. Όλα τα παραπάνω αναλύονται στο παρακάτω στιγμιότυπο οθόνης.

```
import React, {useState,useEffect} from 'react';
import {StyleSheet,Text,View,Image,TextInput,TouchableOpacity,Dimensions,Button,Alert} from 'react-native';
import api from '../api/api';
import axios from 'axios';
import * as SecureStore from 'expo-secure-store';
import moment from 'moment';
import { ScrollView } from 'react-native-gesture-handler';


const BookEvent = (props) => {
    const [ticket, setTicket] = useState(0);

    const getTicket = async () => {
        return ticket;
    };

    useEffect(() => {
        getTickets();
    }, []);
}

const windowWidth = Dimensions.get('window').width;
const windowHeight = Dimensions.get('window').height;
```

Εικόνα 75: Στιγμιότυπο κώδικα του αρχείου event.js 1/3

Στη συνέχεια με την useState ορίζονται και οι εξής μεταβλητές:

- title: Στη συγκεκριμένη μεταβλητή ορίζεται ο τίτλος της εκδήλωσης μόλις ανοίξει η οθόνη.
- eventId: Στη συγκεκριμένη μεταβλητή ορίζεται ο μοναδικός κωδικός της εκδήλωσης μόλις ανοίξει η οθόνη.
- eventData: Στη συγκεκριμένη μεταβλητή ορίζονται όλα τα δεδομένα της εκδήλωσης μόλις ανοίξει η οθόνη.

```
const Event = (props) => {
    const [title,setTitle] = useState('');
    const [eventId,setEventId] = useState('');
    const [eventData,setEventData] = useState([]);

    const getTitle = async () => {
        //const response = props.navigation.getParams('title', 'No event');
        const response = props.route.params.title;
        setTitle(response);
    }

    const getEventId = async () => {
        const response = props.route.params.eventid;
        setEventId(response);
    }

    const getEventData = async () => {
        const eventId = props.route.params.eventid;
        const response = await api.get(`/events/${eventId}`);
        setEventData(response.data);
        console.log(response.data);
    }

    useEffect(() => {
        getTitle();
        getEventData();
        getEventId();

    },[]);
```

Εικόνα 76: Στιγμιότυπο κώδικα του αρχείου event.js 2/3

Κάθε φορά που ανοίγει η συγκεκριμένη οθόνη, αναλόγως την εκδήλωση, εκτελείται ο κώδικας που βρίσκεται εντός της useEffect, δηλαδή εκτελούνται οι συναρτήσεις getTitle, getEventData , getEventId. Με αυτό τον τρόπο επιτυγχάνουμε κάθε φορά να γίνεται μια κλήση στο API και έτσι να έρχονται τα νέα δεδομένα. Αυτό συμβαίνει σε πολύ μικρό χρονικό διάστημα και ο χρήστης δεν καταλαβαίνει σχεδόν τίποτα.

```
return (
    <View style={{flex:1}}>
        <ScrollView>
            <Image
                source = {{uri:`${eventData.img}`}}
                style={{width: windowWidth, height: windowHeight*0.33}}
            />
            <Text style={styles.title}>{title}</Text>
            <Text style={styles.descr}>{eventData.descr}</Text>

            <View style={styles.bookCont}>
                <View style={{alignContent:'center'}}>
                    <Text style={styles.venue}>
                        {moment(eventData.date).format('DD/MM/yyyy'){"  "}{moment(eventData.date).utcOffset('GMT +00:00').format('HH:ss')}
                    </Text>
                </View>
                <View style={{alignContent:'center'}}>
                    <Text style={styles.venue}><Text style={{color:'grey',fontSize:14}}>από</Text>{" "}{eventData.price}€</Text>
                </View>
                <View style={{alignContent:'center'}}>
                    <Text style={styles.venue}>{eventData.venue}</Text>
                </View>
                <TouchableOpacity
                    style={styles.bookBtn}
                    onPress={()=> props.navigation.navigate('Κράτηση',{
                        title: title ,
                        eventid: eventId,
                        evdate: eventData.date
                    })}
                    >

                <Text style={{fontSize:20,color:'white',textAlign:'center'}}>Κράτηση</Text>
                </TouchableOpacity>
            </View>
        </ScrollView>
    </View>
```

Εικόνα 77: Στιγμιότυπο κώδικα του αρχείου event.js 3/3

Το τελευταίο κομμάτι του κώδικα αφορά το UI και το πως προβάλλεται στον χρήστη. Στη συγκεκριμένη οθόνη, το styling είναι inline και αυτό σημαίνει πως όλες οι εντολές CSS βρίσκονται στην εκάστοτε γραμμή στην οποία αναφέρονται. Παρατηρούμε πως μέσα στο γενικό View, υπάρχει το ScrollView ώστε σε περίπτωση αρκετού περιεχομένου, όπως έχουμε αναφέρει παραπάνω, ο χρήστης να μπορεί να κάνει scroll και άρα όλη η πληροφορία να είναι ορατή. Τα στοιχεία συμπληρώνονται στο εκάστοτε σημείο με τον τρόπο που έχουμε δει και παραπάνω, δηλαδή εφόσον τα δεδομένα έχουν αποθηκευτεί σε μια μεταβλητή (στην περίπτωση μας eventData) , στη συνέχεια βρίσκουμε το εκάστοτε πεδίο και το συμπληρώνουμε.

7.6 Alerts

Μια επιπλέον δυνατότητα που προσφέρει η React Native είναι πως έχει εγγενώς το Alert. Δηλαδή μπορεί με πολύ απλό τρόπο (με import δηλαδή) να συμπεριλιφθεί και να εκτελεστεί κατευθείαν. Ο τρόπος με τον οποίο το δημιουργούμε είναι να κάνουμε import τη React και το Alert από τη React Native. Στη συνέχεια χρειάζεται να δημιουργήσουμε μια συνάρτηση και εντός αυτής να ορίσουμε ότι υπάρχει ένα αντικείμενο alert τύπου Alert και να ορίσουμε τι θα εμφανίζεται στο κείμενο του, τι στα κουμπιά του και τι σαν τίτλος.

```
import React from 'react';
import { Alert } from 'react-native'
import {Linking} from 'react-native'

const WrongCredsAlert = () => {

    Alert.alert(
      'Λάθος στοιχεία!',
      'Έλεγξε ξανά τα στοιχεία που έχεις καταχωρήσει!',
      [
        {text: 'OK', onPress: () => console.log('Someone probably needed help...')},
      ],
      {cancelable: false},
    );

}
export default WrongCredsAlert
```

Εικόνα 78: Στιγμιότυπο κώδικα του αρχείου wrongCredsAlert.js

Τέλος με την εντολή export default WrongCredsAlert κάνουμε διαθέσιμο το συγκεκριμένο alert και από άλλες οθόνες, για παράδειγμα από την loginScreen

7.7 Api.js

```
import axios from 'axios';

const instance = axios.create({
    baseURL: 'http://95.216.140.67:3000'
});

export default instance;
```

Εικόνα 79: Στιγμιότυπο κώδικα του αρχείου api.js

Το αρχείο Api.js που υπάρχει στον φάκελο api είναι υπεύθυνο για την δημιουργία ενός instance μέσω της βιβλιοθήκης axios. Αυτό που επιτυγχάνεται με τη συγκεκριμένη βιβλιοθήκη είναι να μπορούν να γίνουν κλήσεις GET & POST μέσα από μια εφαρμογή React Native. Το συγκεκριμένο αρχείο χρειάζεται να έχει τουλάχιστον το baseURL όπως λέγεται, δηλαδή τη διεύθυνση του αρχείου που βρίσκεται το API. Στη συνέχεια, το συγκεκριμένο αρχείο συμπεριλαμβάνεται μέσω της εντολής import, στα αρχεία των οθονών και έτσι υπάρχει η δυνατότητα δημιουργίας κλήσεων από την εκάστοτε οθόνη.

Σχετικά με μερικές οθόνες που παραλήφθηκαν, η παράλειψη έγινε σκόπιμα καθώς υπήρχε συνεχής επανάληψη κώδικα και θα ήταν ανούσιο να αναφερόμαστε σε σχεδόν πανομοιότυπα πράγματα.

7.8 Επίλογος

Φτάνοντας στο τέλος της υλοποίησης της mobile εφαρμογής διαπιστώθηκε ότι η δομή των αρχείων μιας εφαρμογής σε React Native είναι αρκετά απλή και αρκετά φιλική προς τον προγραμματιστή όσον αφορά την υλοποίηση και στα δύο λειτουργικά συστήματα (adnroid & iOS), σε σχέση με άλλες πλατφόρμες που για το κάθε λειτουργικό χρειάζονται διαφορετική υλοποίηση. Δηλαδή με ένα codebase και χωρίς καμία άλλη παρέμβαση, η εφαρμογή εκτελείται ομαλά και ορθά και στα δύο λειτουργικά. Επιπλέον η πληθώρα διαθέσιμων δωρεάν πακέτων (node package manager) που υπάρχουν, βοηθάει στο να μην υλοποιηθεί από την αρχή κάτι το οποίο είναι συχνά χρησιμοποιούμενο. Για παράδειγμα για τη διασύνδεση με το API, υπάρχει διαθέσιμο το πακέτο axios.

Εν τέλει, αυτό που επιτυγχάνεται με την React Native είναι ότι με σχετικά καλές γνώσεις javascript αλλά και γνώσεις React (hooks, lifecycle, state) μπορεί κάποιος να φτιάξει μια εφαρμογή για κινητές συσκευές.

Κεφάλαιο 8ο: Συμπεράσματα

Με τη δημιουργία του API, κατέστη εφικτή η επικοινωνία ανάμεσα στη βάση δεδομένων και τις δύο εφαρμογές. Το μεγάλο πλεονέκτημα του είναι ότι δε χρειάζεται η κάθε εφαρμογή να υλοποιεί τη δική της διασύνδεση με τη βάση δεδομένων, απλώς να χρησιμοποιεί κάποιον ήδη διαθέσιμο τρόπο διασύνδεσης (axios, ajax). Πλέον αυτό θεωρείται δεδομένο στην κοινότητα των προγραμματιστών καθώς τα τελευταία χρόνια οι περισσότερες μεγάλες εφαρμογές έχουν το δικό τους API και κάποιος τρίτος μπορεί εύκολα να διασυνδεθεί.

Πέρα από αυτό, καθοριστικό ρόλο έπαιξε η χρήση του expo στην ανάπτυξη της εφαρμογής για κινητά. Οι δυνατότητες που είχε ώστε να επιτευχθεί πολύ γρήγορα η έναρξη ανάπτυξης της εφαρμογής χωρίς να χρειάζεται κάποια παραμετροποίηση είναι μακράν το μεγαλύτερο πλεονέκτημα του.

Σχετικά με την διαδικτυακή εφαρμογή διαχείρισης εκδηλώσεων, ο τρόπος με τον οποίο μπορεί να προστεθεί μια νέα δυνατότητα είναι εξαιρετικά εύκολος καθώς τα αρχεία είναι δομημένα με τέτοιο τρόπο ώστε να γίνεται εύκολα κατανοητή από ένα τρίτο μάτι και να μπορεί εύκολα να τροποποιηθεί.

Σχετικά με νέες δυνατότητες, η εφαρμογή για κινητά μπορεί να βελτιωθεί προσθέτοντας υπηρεσίες τοποθεσίας (geolocation) ώστε ο χρήστης να βλέπει εκδηλώσεις που βρίσκονται στην πόλη του ή σε κάποια ακτίνα μακριά του. Επιπλέον μπορεί να υπάρξει η προσθήκη ενός payment gateway, όπως για παράδειγμα το stripe ή η viva έτσι ώστε να μπορεί ο χρήστης να ολοκληρώσει την πληρωμή των εισιτηρίων του κατευθείαν μέσα από την εφαρμογή. Αυτό συνδέεται και με μια αντίστοιχη οθόνη στην διαδικτυακή εφαρμογή για διαχείριση των εκδηλώσεων ώστε ο εκάστοτε διοργανωτής να μπορεί να έχει εικόνα των συναλλαγών.

Γενικά, ο σχεδιασμός και η ανάπτυξη εφαρμογών είτε mobile είτε web, αποτελείται από μια συνεχόμενη ροή βελτιώσεων και διορθώσεων. Σε κάθε νέα έκδοση μιας εφαρμογής θα πρέπει να διορθώνονται τυχόν bugs αλλά και να προστίθενται νέες δυνατότητες έτσι ώστε να μπορεί να είναι ανταγωνιστική αλλά και να καλύπτει τις ανάγκες του χρήστη.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Για την υλοποίηση της παρούσας πτυχιακής εργασίας χρησιμοποιήθηκαν οι παρακάτω πηγές:

Βιβλία

[1] Learning React Native: Building Native Mobile Apps with JavaScript, Bonnie Eisenman 2016
[2] Programming PHP: Creating Dynamic Web Pages 3rd Edition, Kindle Edition, Kevin Tatroe 2013
[3] Node.js Web Development - Fourth Edition, David Herron May 2018

Internet Site

[4] Official PHP Website https://www.php.net/
[5] Official Expo Website https://expo.io/
[6] Official React Native Website https://reactnative.dev/
[7] Official Node.js Website https://nodejs.org/
[8] Figma - Mockup Design Tool https://www.figma.com/
[9] RESTFUL Api https://restfulapi.net/
[10] Official MongoDB Website https://www.mongodb.com/
[11] Official Ubuntu Website https://ubuntu.com/